LASER

Home-Computer

Software-System
Handbuch I

SANYO Video Vertrieb, Hamburg

Wolfgang Radeloff

LASER

Home-Computer

Software-System
Handbuch I

SANYO Video Vertrieb, Hamburg

Impressum:

Verlegerische Betreuung und Satz: Roland Löhr, Ahrensburg
Gestaltung des Buchumschlages und grafische Arbeiten: Cathrin Utescher, Hamburg
Druck und Herstellung: Kuncke Druck GmbH, Ahrensburg

Einleitung

Das Ihnen hier vorliegende Buch bietet in 12 Kapiteln und einem umfangreichen Anhang alle die Informationen, die im BASIC-Handbuch der LASER-Computer zu kurz gekommen sind.

Das System hält für den Anwender eine Fülle von weiteren Dienstleistungen bereit, die jetzt mit Kenntnis des Betriebssystems genutzt werden können. Dies sind unter anderem: Rechnen mit 16 signifikanten Stellen, ON...GOTO und ON... GOSUB, FRE(0) und das Auffinden von Variablen im Speicher. Binäre Dateien lassen sich nun abspeichern und starten, der Anschluß von Peripheriegeräten über den USER-Port wird dokumentiert.

Im übrigen beantwortet das Buch die Fragen, die dem Verfasser von seiten der Anwender immer wieder am Telefon gestellt wurden. Zusammen mit dem BASIC-Handbuch ist so schon ein besserer Stand der Dokumentation und Handhabung erreicht. Noch offene Fragen können in einem folgenden Buch behandelt werden.

Zur Orientierung:

Bevor Sie mit diesem Buch arbeiten, sollten Sie die folgenden drei Punkte stets beachten:

1. Es existieren für den hier beschriebenen Computer (LASER 110/210/310 und VZ 200) zwei Software-Versionen: LASER-BASIC Vers. 1.1 und Vers. 2.0. Der gravierende Unterschied besteht im Aufbau des Bildes im Video-RAM. Vers. 1.1 beschreibt das RAM mit normalen Zeichen, während Vers. 2.0 den Bildschirm mit inversen Zeichen initialisiert. Das Buch bezieht sich stets auf die Vers. 1.1.

   Sollten Sie die BASIC-Version 2.0 in Ihrem Gerät haben, so halten Sie während des Einschaltens die CTRL-Taste gedrückt, der Bildschirm wird dann wie mit Vers. 1.1 arbeiten, und alle Beispiele hier im Buch werden einwandfrei arbeiten.

2. Es wird in diesem Buch viel auf die BASIC-Erweiterung 'BASIC-UP" Bezug genommen. Wenn Sie diese Kassette besitzen, sollten Sie wissen, daß mit 'BASIC-UP" nur BASIC-Programme arbeiten, die mit "BASIC-UP" erstellt wurden.

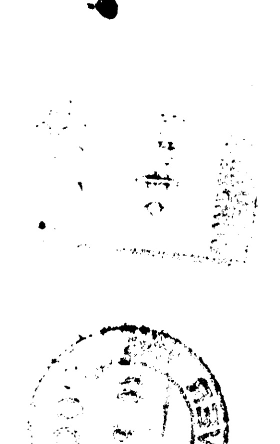
3. "EXTENDED BASIC", ein weiteres BASIC-TOOLKIT, ergänzt 'BASIC-UP" mit einem umfangreichen Grafik-Befehlssatz. CIRCLE, RECT, PAINT, PLOT.. TO..., GCLS und LPEN für den Lichtgriffel sind nur einige Beispiele.

Das Buch ist im übrigen so gegliedert, daß die einzelnen Kapitel in sich abgeschlossen sind und ohne Rücksicht auf die Reihenfolge durchgearbeitet werden können.

Hamburg, im Juli 1984

Inhaltsverzeichnis

Verzeichnis der Anhänge



1 Zahlen, Zeichen, Variable

Zahlen- und Zeichenformate
Variablenarten

LET, DEFINT, DEFSNG, DEFDBL

Ihr LASER-Computer verarbeitet Zahlen, die sich in normaler Schreibweise nicht mehr ausdrücken lassen. Der Zahlenbereich ist, als Zehnerpotenzen ausgedrückt, größer als $1*10^{-39}$ und kleiner als $1*10^{39}$. In den Computer werden solche Zahlen eingegeben in der Form: 1E38 oder 1E-38. Die Zahl 1812 wird in dieser technisch-wissenschaftlichen Darstellung als 1.812E3 geschrieben. Natürlich versteht der Computer auch die normale Schreibweise. Wenn er jedoch eine Zahl gleich oder größer als 999999.5 ausgeben soll, so schaltet er im Normalfall auf das technisch-wissenschaftliche Format um (1.2). Ist die Zahl kleiner als 0.0099, so wird ebenfalls auf das technisch-wissenschaftliche Format umgeschaltet.

```
? 2.33 E 3 <RETURN>
  233Ø
                                   1.1
```

```
? 3470828 <RETURN>
  3.47083 E+06
                                   1.2
```

Single Precision

Diese Ausgabe erfolgt mit sechs Ziffern, die restlichen Stellen werden gerundet (1.3).

Diese Art der Zahlendarstellung wird als 'Fließkommazahl, einfache Genauigkeit' bezeichnet (engl. Floating Point, Single Precision). Nach dem Einschalten befindet sich der Rechner in dieser Darstellungart. Bei allen Berechnungen ist besonders der Rundungsfehler zu beachten!

Ist der Rundungsfehler nicht akzeptabel und wird eine größere Genauigkeit des Rechenergebnisses notwendig, so kann auf 'doppelte Genauigkeit' (Double Precision) umgestellt werden. Die Zahlen werden zur Eingabe oder im Programm technisch-wissenschaftlich definiert, jedoch ist das 'E' durch ein 'D'

zu ersetzen (1.4). Die Zahlendarstellung erfolgt jetzt mit 16 signifikanten (gültigen) Ziffern, und der Rundungsfehler wird für die meisten Anwendungen vernachlässigbar klein.

```
? Ø.1234567 <RETURN>
  .123457
```
1.3

```
? 1/6 D Ø
  .1666666666666667
```
1.4

```
VARIABLEN-TYP-TABELLE
=====================
30977........A
30978........B
  .
  .
31001........Y
31002........Z
```
1.5

Variablen-Speicher

Bisher haben wir die Zahlendarstellung direkt als 'Konstanten' betrachtet. Im Programm werden Zahlen in Speichern abgelegt. Dort können sie im Programmlauf geändert und manipuliert werden. Diese so abgespeicherten Zahlen werden als 'Variable' bezeichnet. Solche Zahlenspeicher, numerische Variable genannt, werden vom Programm ohne besondere Vorkehrungen stets für Single Precision-Zahlen angelegt. Um Zahlen im Format doppelter Genauigkeit abzuspeichern, ist ein Eingriff in die Variablenhandhabung des BASIC-Interpreters notwendig: Ab der Adresse 30977 hat der Interpreter eine Tabelle von 26 Bytes Länge angelegt. Jeder Tabellenplatz entspricht einem Buchstaben des Alphabets (1.5). Durch Eintragen von Kennzahlen (1.6) mit Hilfe der POKE-Anweisung

```
FLAG VARIABLEN-TYP
==================
2 = INTEGER-ZAHLEN
3 = ZEICHENKETTEN
4 = SINGLE PRECISION
8 = DOUBLE PRECISION
```
1.6

lassen sich Variable gruppenweise für einen entsprechenden Variablentyp erklären (deklarieren).

Stringvariable und Integervariable können im Programm nun unter Weglassung des Typkennzeichners angesprochen werden. Ausdrücke wie Y = 12 statt Y% = 12 oder A="ZEICHENKETTE" statt A$ = "ZEICHENKETTE" sind dann möglich.

Die Bezeichner bleiben aber vorrangig. Ist die Variablengruppe B für die Aufnahme von Intergerzahlen definiert, so wird der Stringspeicher BA$ trotzdem vorrangig für die Aufnahme von Zeichenketten angelegt.

```
10 POKE 30978,8
20 BZ = 1/6D
30 PRINT BZ
RUN
.1666666666666667
                                1.7
```

Variable doppelter Genauigkeit

Beispielsweise werden mit POKE 30978,8 alle im Namen mit 'B' anfangenden Variablen für die Aufnahme von Zahlen im 'Double Precision'-Format eingerichtet. Das sind Namen wie B, B9, BZ, B(5) oder BILD.

Beachten Sie aber, daß Zuweisungen an Variable für doppelte Genauigkeit im technisch-wissenschaftlichen Format mit dem 'D' als Exponentenkennzeichen erfolgen müssen (1.7)! Wird im Beispiel (1.7) das D (D0) weggelassen, so ist das falsche Ergebnis 1/6 = .1666666716337204.

Operationen mit doppelter Genauigkeit sollten im Programm jedoch nur dort durchgeführt werden, wo die erhöhte Genauigkeit des Ergebnisses erforderlich ist. Das Beispiel (1.8) braucht für 100 Multiplikationen doppelter Genauigkeit ca.

```
10 POKE 30979,8: 'DBL
20 C1 = 1.5 D
30 FOR I=1 TO 100
40 C2 = C1 * 1.234
50 NEXT
60 PRINT C2

**** 12 SEKUNDEN ****
                                1.8
```

```
10 C1 = 1.5
20 FOR I=1 TO 100
30 C2 = C1 * 1.234
40 NEXT
50 PRINT C2

**** 2 SEKUNDEN ****
```

1.9

12 Sekunden, mit Single Precision jedoch nur 2 Sekunden (1.9). Außer der längeren Laufzeit muß für Variable doppelter Genauigkeit als Nachteil auch der erhöhte Speicherraum in Kauf genommen werden (1.10).

```
INTEGER      :  5 BYTES
SINGLE PREC.:  7 BYTES
DOUBLE PREC.: 11 BYTES
STRING       :  6 +(LÄNGE) BYTES
```

1.10

```
A% = 12
```

1.11

Integer-Variable

Ein drittes Variablenformat läßt nur ganzzahlige Zahlen im Bereich von -32768 bis +32767 zu. Kleiner Speicherbedarf und schnelle Bearbeitung sind der Vorteil gezielter Anwendung dieser 'Integer-Variablen'. Solche Variablen können mit Hilfe der Tabellen (1.5) und (1.6) oder durch Kennzeichnung des Variablennamens mit einem nachgestellten '%' eingerichtet werden (1.11).

Wandeln des Formats

Variable können von einem Format zum anderen gewandelt werden. Jedoch sind Einschränkungen zu beachten:

* Fließkomma zu Integer: Nachkommastellen werden abgeschnitten. Es erfolgt keine Rundung. Beispiel: A% = A (1.12).

* Single Precision zu Double Precision: Ein vorher eingetretener Rundungsfehler wird nicht rückgängig gemacht. Beispiel: POKE 30978,8: B = A .

* Double Precision zu Single Precision: Es wird gerundet. Beispiel: POKE 30978,8 A = B .

* Eine Wandlung vom Gleitkomma-Format nach Integer ist auch mit dem Ausdruck A = INT(B) möglich.

```
10 A = 12.25
20 A% = A
30 PRINT A%
RUN
  12
                    1.12
```

BASIC UP

Mit der BASIC-Erweiterung 'BASIC UP' entfällt die Einstellung des Variablentyps über die Tabelle im System-RAM. Statt dessen stehen die Anweisungen:

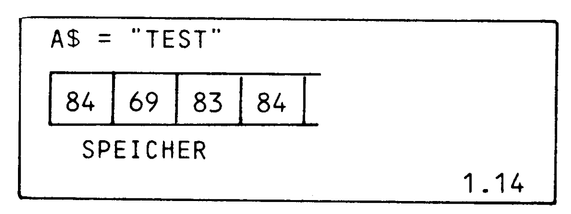
```
■DEFINT
■DEFSNG
■DEFDBL
```

für den Initialisierungsteil des Programmes zur Verfügung. Eine Liste der Variablen-Anfangszeichen ist anzuhängen. Alle Variablen, die dann diese Zeichen als erste im Namen führen, sind so für den entsprechenden Typ initialisiert (1.13). Eine Kennzeichnung der Integer-Variablen mit '%' im Programmtext kann entfallen, wenn über die Tabelle oder 'BASIC UP' eine entsprechende Vereinbarung getroffen wurde.

```
NEUE ANWEISUNGEN
================
■DEFINT A,W,Z
■DEFSNG B,C,M
■DEFDBL D,E,M-P
                    1.13
```

Zeichenketten - Strings

Der vierte Variablentyp: Die Zeichenkette oder 'STRING'. Dieser Variablenart können Zeichenketten (Ziffern, Buchstaben, Zeichen und Grafikzeichen), invers oder normal dargestellt, unter ihren Kodezahlen zur Speicherung zugewiesen werden. Diesen Variablen wird zur Kennzeichnung im Namen ein '$'-Zeichen (Dollar) zugeteilt. Auch diese Stringvariablen lassen sich über die Tabelle vordefinieren. Ihre Kennzahl ist '3'. Informieren Sie sich über die Zeichencodes im Anhang 13.



1.14

BOOL'sche Variable

Dieser Variablentyp sei als letzter erwähnt. In ihm wird das Ergebnis eines Vergleiches oder einer Verknüpfung zweier oder mehrerer Variablen festgehalten. Man nennt ihn auch 'Wahrheitsvariable'.

Dieser Variablentyp wird normalerweise durch den Ausdruck nach IF... erzeugt und mit den Anweisungen nach THEN... und ELSE... sofort ausgewertet. Für spezielle Anwendungen läßt er sich einer numerischen Variablen (Integer) zuweisen und in späteren IF...THEN-Anweisungen auswerten.

```
BOOL'SCHE VARIABLE
==================
WAHR  :   -1 BZW. ≠ Ø
FALSCH:    Ø
```
1.15

```
10 A=11: B=12: C=12: D=13
20 X= B=C: Y= A>B: Z= A<D
30 PRINT X;Y;Z
RUN
-1  Ø  -1
```
1.16

Beispiel 1.17: Eine Variable wird auf Null getestet. Das Programm verzweigt, wenn A ungleich Null ist.

Beispiel 1.18: Das Programm verzweigt, wenn A gleich 0 ist.

```
IF A THEN GOTO 5ØØ
```
1.17

```
IF NOT A THEN GOTO 500
```
1.18

2 Felder: Organisierte Speicher

Feldvariable: Listen, Tabellen und Karteien

DIM

Alle im Kapitel 1 angesprochenen Variablentypen lassen sich zu Speicherfeldern organisieren. Das Feld (engl. Array) bekommt einen Namen (2.1). Der einzelne Speicherplatz innerhalb des Feldes wird durch einen in Klammern gestellten Index angesprochen (2.2). Dieser Index kann eine Konstante, eine numerische Variable oder ein numerischer Ausdruck sein (2.3).

Der Vorteil dieser Organisation: Speicherplätze werden nicht mehr durch direkte Nennung ihres Namens angesprochen, sondern durch computergesteuerte Operationen. Beispiel 2.4 gibt den Inhalt von 10 Variablen auf den Bildschirm aus.

```
Feldvariable:  Namen
====================
AB, D$, E1%
                                   2.1
```

```
Feldvariable mit Index
======================
AB(5), D$(Ø), E1%(12)
                                   2.2
```

```
Feldvariable mit Index
Konstante, Variable, Ausdruck
=============================
AB(5), D$(J), E1%(INT(J/3))
                                   2.3
```

```
1Ø FOR I=1 TO 1Ø
20 PRINT AB(I)
3Ø NEXT I
                                   2.4
```

Wird in einem Programm einer indizierten Variablen mit LET ein Wert zugewiesen, so legt der LASER automatisch ein Speicherfeld mit 11 Plätzen an und stellt den mit LET zugewiesenen Wert in den entsprechenden Speicherplatz ein (2.5). Einerseits bedeutet das für Felder mit weniger als 11 Speicherplätzen eine nicht wirtschaftliche Speicherverwaltung (einige Plätze werden nicht belegt), andererseits muß mit dem BASIC-

```
10 LET AB(5)=12.5
RUN
AB(0) = 0
AB(1) = 0
..
AB(5) = 12.5
..
AB(10) = 0
                                  2.5
```

Interpreter für Felder mit mehr als 11 Plätzen eine Zuweisungsvereinbarung getroffen werden, so daß aus Gründen der Übersichtlichkeit des Programmes auf die automatische Anlage bei kleinen Feldern verzichtet werden sollte.

Statt dessen wird mit dem BASIC-Schlüsselwort DIM (DIMensioniere) für jedes Feld die Speicherorganisation in ihrer Größe definitiv festgelegt. Die Anweisung (2.7) legt unter dem Variablennamen AB 16 Speicherplätze, beginnend mit AB(0) bis AB(15) und dem Wert Null, an.

Mehrere Felder können mit einer Anweisung initialisiert werden. Die benannten Felder werden mit ihrer Dimensionierung in einer Liste, durch Kommas getrennt, nach dem Wort DIM angegeben (2.8). Die DIM-Anweisung muß im Programm stets vor Zuweisung von Werten an die Feldvariablen stehen. Sie gehört also in den Initialisierungsteil des Programmes. Ein Feld darf im Verlauf des Programmes nur einmal mit DIM definiert werden. Der Versuch, ein Feld unter dem gleichen Namen mit anderer Ausdehnung zu dimensionieren, wird mit einer Fehlermeldung und einem Abbruch des Programmlaufes verhindert. (2.9).

```
DIMensioniere
                                  2.6
```

```
10 DIM AB(15)
                                  2.7
```

```
10 DIM AB(15),D$(5),E1%(20)
                                  2.8
```

```
1Ø DIM A$(4)
2Ø DIM A$(5)
RUN
? REDIM'D ARRAY ERROR IN 2Ø
                      2.9
```

Eine weitere Fehlermeldung des Betriebs-Systems wird immer dann ausgegeben, wenn auf Feldvariable zugegriffen wird, deren Index außerhalb des dimensionierten Bereiches liegt (2.10).

```
10 DIM A(5)
20 PRINT A(12)
RUN
? BAD SUBSCRIPT ERROR IN 20
                        2.10
```

Tabellen und Karteien

Bisher wurden nur Felder in Form einer Liste betrachtet (2.5). Der LASER erlaubt zwei weitere Organisationsformen: Das sind Felder in Tabellenform, die eine Anordnung der Variablen nach Zeile und Spalte ermöglichen, und Felder in der Form einer Kartei, bei der jede Karte wiederum in Tabellenform angelegt ist.

Das Feld in Tabellenform, auch 'zweidimensionales Feld' oder 'Matrix' genannt, wird nach dem Feldnamen mit zwei Indizes angesprochen. Der erste benennt die Zeile der Tabelle, der zweite die Spalte (2.11 und 2.12). Dementsprechend erfordert die Organisation in Karteien die Angabe von drei Indexzahlen: Spalte und Zeile der Tabelle und Nummer der Karte. Die Speicherorganisation erstreckt sich hier in drei Dimensionen (2.13, 2.14).

E1%

Ø,Ø	Ø,1	Ø,2	Ø,3
1,Ø	1,1	1,2	1,3
2,Ø	2,1	2,2	2,3
3,Ø	3,1	3,2	3,3

2.11

```
1Ø DIM E1% (3,3)
20 E1%(1,2)=12
30 PRINT E1%(1,2)
RUN
12
```

2.12

```
10 DIM Z%(3,3,3)
20 Z%(1,0,1)=12
30 PRINT Z%(1,0,1)
RUN
12
```

2.13

```
Z%
                  0,0,3 0,1,3 0,2,3 0,3,3
            0,0,2 0,1,2 0,2,2 0,3,2 ,3
      0,0,1 0,1,1 0,2,1 0,3,1 ,2  ,3
0,0,0 0,1,0 0,2,0 0,3,0 ,1  ,2  ,3
1,0,0 1,1,0 1,2,0 1,3,0 ,1  ,2
2,0,0 2,1,0 2,2,0 2,3,0 ,1
3,0,0 3,1,0 3,2,0 3,3,0
```

2.14

Grenzen nur durch den verfügbaren Speicherraum

Es kann pro Feld eine sehr hohe Anzahl von Elementen programmiert werden, die einerseits durch den verfügbaren Speicherraum beschränkt wird, zum anderen läßt aber das LASER-BASIC nur eine bestimmte Ausdehnung der Felder zu. Die Begrenzung für die entsprechenden Variablentypen sind der Tabelle 2.15 zu entnehmen. Sie wurden für den LASER 110 ermittelt. Die Werte in dieser Tabelle beschreiben nur die in der DIM-Anweisung höchste zulässige Dimensionierung, eine Fehlermeldung wird je nach Speichergröße früher auftreten.

DIM dimensioniert alles!!

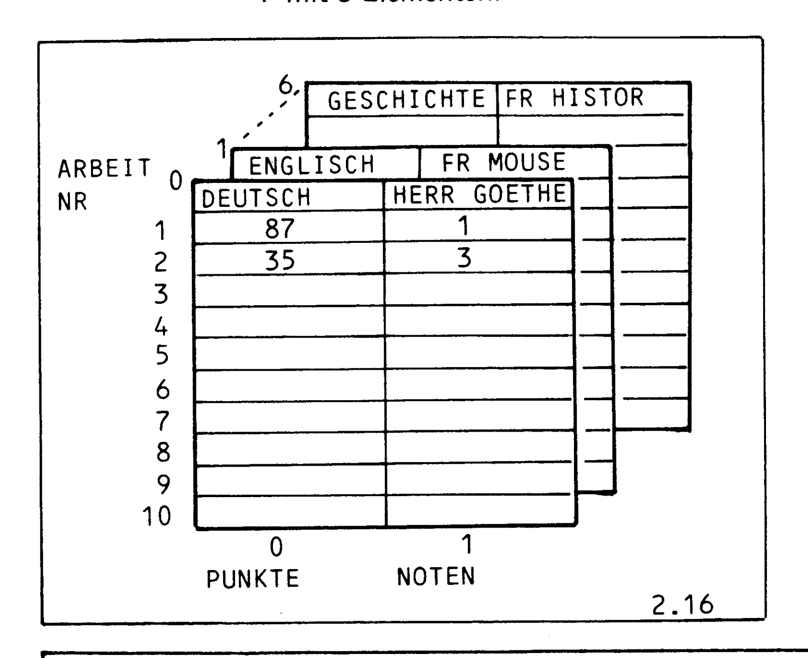
Die DIM-Anweisung gestattet nicht nur die Anlage von Feldern in der beschriebenen Art, sondern legt auch die gesamte Variablentabelle durch Nennung des Variablennamens nach DIM an. (Siehe hierzu auch Kapitel 3 und Anhang 9.) Es wird damit möglich, im Initialisierungsteil des Programmes die Variablentabelle sinnvoll anzulegen und damit Einfluß auf die Verarbeitungsgeschwindigkeit des Interpreters zu nehmen.

DIM A, Z%, C1$, F(5)

Tabelle 2.15 Begrenzung der Feldelemente

	INTEGER	SNG.PREC.	DBL.PREC.	STRING
LISTE	17000	8509	4250	11347
TABELLE	129*129	91*91	64*64	105*105
KARTEI	23*23*23	19*19*19	15*15*15	21*21*21

Diese Anweisung legt die Variablen A und Z% als erste und zweite mit dem Wert Null in der Variablentabelle an, als dritte dann die Stringvariable C1$ und dimensioniert dann das Feld F mit 6 Elementen.



2.16

```
10 'INITIALISIERUNG UND MENUE
20 CLEAR 1000: DIM NO$(10,1,6)
40 RESTORE: CLS
50 FOR I=0 TO 6: READ NO$(0,0,I),NO$(0,1,I)
60 : PRINT I+1;"=";NO$(0,0,I),NO$(0,1,I)
70 NEXT: PRINT
100 PRINT"WELCHES FACH ";: INPUT K
110 IF K<1 OR K>7 THEN PRINT CHR$(27);CHR$(27): GOTO100
115 K=K-1
120 PRINT
121 PRINT" 1 = EINGABE"
122 PRINT" 2 = AUSGABE"
129 PRINT"IHRE WAHL";: INPUT L
```

```
130 IF L<1 OR L>2 THEN PRINT CHR$(27);CHR$(27): GOTO129
140 IF L=1 THEN 300
141 IF L=2 THEN 600
300 'EINGABE
310 CLS: PRINT NO$(0,0,K),NO$(0,1,K): PRINT
320 PRINT"   ARBEIT NR. "
330 PRINT"   ANZ. PUNKTE "
340 PRINT"   NOTE        "
350 PRINT@85,"";: INPUT I
355 IF I>10 THEN 350
360 PRINT@117,"";: INPUTP$
370 PRINT@149,"";: INPUT N$
380 NO$(I,0,K)=P$: NO$(I,1,K)=N$
390 PRINT: PRINT"MENUE (JA/NEIN) ";: INPUTK$
400 IF LEFT$(K$,1)="N" THEN 300 ELSE 40
600 'AUSGABE
610 CLS: MP=0: MN=0: Z=0
620 FOR I=0 TO 10: PRINT NO$(I,0,K),NO$(I,1,K): GOSUB 900: NEXT
630 Z=Z-1: IF Z=0 THEN 800 ELSE PRINT MP/Z,MN/Z: GOTO800
800 'RUECKKEHR ZUM MENUE
810 PRINT" MENUE (JA/NEIN) ";: INPUT K$
820 IF LEFT$(K$,1)="J" THEN 40 ELSE END
900 'MITTELWERTE BILDEN
910 IF NO$(I,0,K)<>"" AND NO$(I,1K)<>"" THEN Z=Z+1
920 MP=MP+VAL(NO$(I,0,K)): MN=MN+VAL(NO$(I,1,K))
990 RETURN
1000 DATA DEUTSCH,HERR GOETHE,ENGLISCH,FR MOUSE
1020 DATA FRANZOES,FRL PARIS,BIOLOGIE,HERR BAUM
1040 DATA PHYSIK,HERR OHM,MATHE,HERR A RIESE
1060 DATA GESCHICHTE,FR HISTOR
```

2.17

Das Programm 2.17 benutzt zum Ablegen von 10 Zensuren in 7 Fächern ein dreidimensionales Feld (Kartei) (2.16). Sehen Sie sich besonders die Handhabung des Feldes an:

Zeile 20 dimensioniert das Feld.

Zeile 50 liest in die Zeile 0 jeder Tabelle die Überschrift ein.

Zeile 310 gibt die Überschrift der gewählten Tabelle aus.

Zeile 380 liest Werte in eine definierte Zeile ein.

Zeile 620 gibt den Inhalt einer Tabelle aus.

Zeile 910 prüft, ob eine Zeile beschrieben ist.

Zeile 920 addiert die Inhalte einer Zeile auf.

3 65536 Speicher werden verwaltet

EPROM-Anwender-Programme, Zeiger im Betriebssystem; mit NEW gelöschte Programme wieder herstellen; Free Bytes

CLEAR, NEW, FRE (0), FRE (""), MEM, VARPTR

Die LASER-Computer (110, 210, 310, VZ 200) sind als Z-80 Systeme organisiert. Anhang 10a gibt eine Übersicht über die physikalische Speicherverteilung des Computers. Anhang 10c ergänzt sie mit der Lage der Speichererweiterungen. Verwaltet werden diese Speicherplätze vom Betriebssystem und dem BASIC-Interpreter, die selber 16 KB Speicherraum belegen (ROM 1, ROM 2).

```
        NAME GAME
STACK   EQU 7FFFH
CRTDGE  EQU 4000H
        ORG 787DH
INTVTR  DEFS 3          ;INTERRUPT EXIT LOC
;
;
        ORG CRTDGE
        DEFB 0AAH       ;1ST PATTERN
        DEFB 055H       ;2ND PATTERN
        DEFB 0E7H       ;3RD PATTERN
        DEFB 018H       ;4TH PATTERN
START   LD SP,STACK     ;RE-INITIALIZE STACK
        LD A,0C3H       ;JP OBJ CODE
        LD (INTVTR),A
        LD HL,INTSVC    ;GET SERVICE ROUTINE ADDR
        LD (INTVTR÷1),HL        ;MODIFY EXIT LOC
            .
            .
            .
INTSVC  CALL KEYBRD     ;USER WRITTEN INT SERVICE ROUTINE
            .
            .
        POP HL          ;CLEAR STACK
        POP HL          ;RECOVER HL REG
        POP DE          ;RECOVER DE REG
        POP BC          ;RECOVER BC REG
        POP AF          :RECOVER AF REG
        EI              ;ENABLE INTERRUPT
        RETI
```

3.1

EPROM-Programme

Nach dem Einschalten prüft das Betriebssystem, ob ab den Adressen 4000H, 6000H oder 8000H die Bitmuster AAH, 55H, E7H und 18H vorhanden sind. Werden sie in einem dieser Bereiche festgestellt, so übergibt das Betriebssystem die Kontrolle an das nach diesen vier Bitmustern beginnende Anwenderprogramm. Programm 3.1 ist ein Beispiel zur Einbindung eigener Programme.

Zeiger im Betriebssystem

Der BASIC-Interpreter verwaltet seinen RAM-Bereich (7AE9H bis FFFFH) über eine Anzahl von 'Zeigern', die zusammen mit anderen Systemvariablen nach dem Einschalten im RAM-Bereich (7800H bis 7AE8H) angelegt werden. Siehe Anhang 11.

Zeiger sind zwei Bytes breite Register, in denen sich der BASIC-Interpreter Adressen der Daten- und Programmtextbereiche merkt, die er zur Verwaltung braucht.

Für den BASIC-Programmierer sind die Zeiger HP (Beginn des BASIC-Textes) und TOM (Zeiger auf die letzte RAM-Zelle) von besonderem Interesse. Ein Umsetzen dieser Zeiger schafft vor BASIC geschützten Speicherraum zur Ablage von Maschinen-Programmen. Üblicherweise werden bei den Z-80 BASIC-Versionen diese Programme durch das Umsetzen des Zeigers TOM (Top Of Memory) geschützt. Lesen Sie hierzu auch Kapitel 15!

CLEAR

Die Anweisung CLEAR löscht alle Variablen und dimensionierten Felder. Die Reservierung wird rückgängig gemacht. CLEAR ist ohne Parameter zu geben. Reservierungen für Zeichenketten werden nicht beeinflußt.

```
10 CLEAR 1000
```
3.2

```
PRINT PEEK (30898)
```
3.3

Anhang 11 zeigt den Speicherbereich zur Ablage von Zeichenketten und den dazu gehörenden Zeiger STSP (String Space). Nach dem Einschalten des Computers sind für die Ablage von Zeichenketten 50 Bytes reserviert. Mit der Anweisung CLEAR

und dem entsprechenden Parameter kann der BASIC-Programmierer diesen String-Bereich für die Bedürfnisse seines Programmes einrichten (3.2).

Soll das Programm in allen möglichen Kombinationen der LASER-Computer und Speichererweiterungen laufen und wenn maximaler Platz für die Ablage von Zeichenketten gefordert ist, dann bietet sich der Zeiger TOM als Maß für die Größe des mit CLEAR zu dimensionierenden Stringspeicherbereiches an.

Das MSB (Most Significant Byte - höherwertiges Byte) des Zeigers enthält die Seitennummer der höchsten, beim Speichertest festgestellten RAM-Zelle. Mit der Anweisung (3.3) ist die für das System zutreffende Seitenzahl zu ermitteln (3.4).

Soll das Programm universell seinen String-Speicherbereich selbst dimensionieren, so kann die Programmzeile (3.5) verwendet werden. Die darin genannte Konstante soll den Speicherbedarf für die Geräte-Kombination mit dem geringsten RAM-Bereich angeben.

```
78B2H      30898    (MSB - TOM)
==============================
LASER 110     7FH   127
VZ 200        7FH   127
LASER 210     8FH   143
L110 + 16KB   BFH   191
VZ200+ 16KB   BFH   191
L210 + 16KB   CFH   207
LASER 310     B7H   183
LASER310 + 16KB
              FFH   255
64KB          FFH   255
                          3.4
```

```
20 CLEAR 1000+(PEEK(30898)-127)*256
                              3.5
```

NEW

Anhand der Grafik im Anhang 11 ist leicht festzustellen, was die Anweisung NEW bewirkt: Sie setzt alle Zeiger auf den Wert nach dem Einschalten des Computers zurück. Das Programm ist dann zwar noch vorhanden, der BASIC-Interpreter hat es aber gewissermaßen 'vergessen'.

Ein versehentlich gelöschtes Programm kann durch einige POKE-Anweisungen dem Interpreter wieder in 'Erinnerung' gebracht werden. Dafür ist nur die Kenntnis notwendig, wie die Textablage organisiert und verwaltet wird. Anhang 16 zeigt dies in grafischer Form.

Jede Anweisungszeile beginnt mit einem '0'-Byte. Darauf folgt ein zwei Bytes breiter Zeiger auf den Anfang der nächsten Zeile. Anschließend wird die Zeilennummer in zwei Bytes abgespeichert, gefolgt vom BASIC-Text der Zeile. Alle Schlüsselworte werden umkodiert in ein Byte, das Werte > 127 erhält und in die Speicher eingetragen. Diese codierten Bytes nennt man auch 'tokens'. Alle anderen Zeichen werden mit ihrem ASCII-Code abgelegt, der immer kleiner als 128 ist.

Eine Tabelle der Schlüsselworte und ihrer Codezahlen ist im Anhang 17 zu finden. Anhang 13 gibt den ASCII-Code wieder. Nach der ersten Anweisungszeile folgt wieder ein '0'-Byte als Beginn der zweiten Zeile. Das Ende des BASIC-Textes wird markiert, indem die zwei Bytes, auf die der Zeiger der letzten Zeile zeigt, auf '0' gesetzt werden. Ein BASIC-Programmtext wird im Speicher also immer mit einer Folge von dreimal '0' abgeschlossen.

Ist kein Programm im Speicher vorhanden, also nach NEW, folgen auf die '0' in Zelle 74E8H (Markierung erste Zeile) noch zweimal '0' (3.6). Der Zeiger TP (Ende BASIC-Text/ Anfang Variablen-Tabelle) zeigt nach NEW auf das erste Byte nach den drei Null-Bytes.

Um ein gelöschtes Programm zu restaurieren, ist also nur der Zeiger im BASIC-Text auf die zweite Zeile und der Zeiger TP auf das erste Byte nach den drei Null-Bytes am Ende des früheren BASIC-Textes zu richten. Nachfolgend soll dies mit

```
NACH NEW:
=========

  0     0     0

31464 31465 31466
```

3.6

```
10 A=12
20 PRINTA
```

3.7

dem kurzen Programm 3.7 versucht werden. Das Programm sollte eingegeben werden, und dann sollte mit der Anweisung (3.8) der im Speicher erzeugte BASIC-Code sichtbar gemacht werden. Er entspricht dem im Anhang 16 dargestellten Code. Besonders sind die Werte der Zeiger zu beachten. Der Zeiger der ersten Zeile im Text zeigt auf das erste Byte des Zeigers der zweiten Zeile. Alle Zeilen des BASIC-Programmes sind so miteinander verkettet, und zur Abarbeitung des Programmes 'hangelt' sich der Interpreter anhand dieser Zeiger durch den Text. Der Zeiger TP zeigt auf das erste Byte nach Textende.

```
FOR I=0TO18:?PEEK(31464+I);:NEXT
```

3.8

Mit NEW gelöschtes Programm wiederherstellen

Das Programm (3.7) wird nun mit 'NEW' gelöscht. Aus dem Aufbau des BASIC-Textes kann auf die ungefähre Lage des Zeigers der zweiten Zeile geschlossen werden. Mit PEEK-Anweisungen von der Tastatur her ist seine Adresse nun genau zu ermitteln. Ist die '0', mit der der Beginn der zweiten Zeile markiert wird, gefunden, so steht fest, daß der gelöschte Zeiger der ersten Zeile auf die Zelle 31474 zeigen muß (3.9). Die so ermittelte Adresse ist nun in den niederwertigen und den höherwertigen Teil zu zerlegen (3.10). Anschließend werden die beiden so ermittelten Dezimalzahlen mit Hilfe der POKE-Anweisung in die beiden Zellen des Zeigers eingetragen (3.11). Damit ist der durch NEW gelöschte Zeiger wiederhergestellt.

```
?PEEK (31472)
50
?PEEK (31473)
0
```

3.9

```
? INT (31474/256)
122
? 31474-(122*256)
242
```

3.10

```
POKE 31465,242

POKE 31466,122
```

3.11

Der nächste Schritt ist nun die Restaurierung des Zeigers TP auf das Textende. Auch hier muß die vermutete Lage des Textendes im Speicher mit PEEK-Anweisungen ermittelt werden. Keinesfalls dürfen irgendwelche Schleifen zum Durchsuchen der Speicher formuliert werden, da die damit verbundene Anlage von Variablenspeichern den BASIC-Text weiter zerstören würde. Ist das Textende (dreimal '0') gefunden, kann der Zeiger unter der Adresse 30969, 30970 wiederhergestellt werden. Der Vorgang ist der gleiche wie in (3.10, 3.11). Zu beachten ist, daß die PEEK-Anweisung als Argument ab Adresse 32760 negative Integer-Zahlen haben muß. Ein Konvertierungsprogramm zeigt (4.7).

Free Bytes

Das LASER-BASIC kennt die Anweisung FRE (0) zur Errechnung des noch freien RAM-Speicherraumes nicht. Aus den Werten der Zeiger, die den RAM-Bereich verwalten, ist diese Information aber abzuleiten. Anhang 11 zeigt, daß die Differenz zwischen den Zeigern FSL und STSP dem freien RAM-Bereich in etwa entspricht. Das kurze Programm (3.12), angehängt oder eingebunden in das in Bearbeitung befindliche Programm, wird nach Aufruf die Anzahl der noch freien Bytes ausgeben. Der Stackbereich zur Laufzeit des Programmes wird nicht erfaßt.

Die Routine zur Errechung des freien Speicherraumes ist im BASIC-Interpreter vorhanden, lediglich das entsprechende Schlüsselwort wird nicht erkannt und führt zur Ausgabe einer '0', da FRE als Variablenname interpretiert wird (3.13).

```
100 ST=PEEK(30881)*256+PEEK(30880)
110 FS=PEEK(30974)*256+PEEK(30973)
120 PRINT "FREE=";ST-FS
```
3.12

```
? FRE (Ø)
Ø
```
3.13

```
500 PRINTPRINT(Ø)
```
3.14

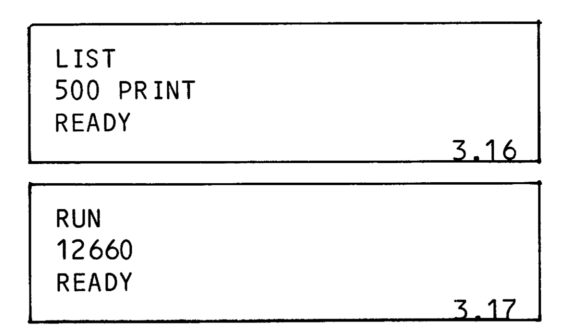
```
POKE 31470,218
```
3.15

FRE (0)

Die Funktion läßt sich aber auf folgende Weise aktivieren: Es wird eine Dummy-Zeile als erste Zeile des Programmes eingegeben (3.14). Anhang 16 zeigt den Aufbau der ersten Zeile. Die beiden PRINT-Anweisungen werden als Tokens mit der Codezahl 178 in den beiden RAM-Zellen 31469 und 31470 erscheinen. Wird das Token in 31470 durch das Token für FRE ersetzt, so wird der BASIC-Interpreter die Zeile als PRINT FRE (0) lesen und korrekt ausführen.

Das Token (verschlüsseltes Befehlsbyte im Speicher) für FRE ist 218. Die Anweisung (3.15) ersetzt das Token 'PRINT' durch das von 'FRE'. Die Zeile wird nach dieser Manipulation nicht mehr korrekt gelistet, aber ausgeführt (3.16, 3.17). Das Anwenderprogramm kann nun geschrieben werden. Alle Editierfunktionen werden die Zeile 500 berücksichtigen. Andere Zeilen können vor und nach dieser Zeile angeordnet werden.

Die Schreibweise FRE (0) der Funktion liefert als Resultat den noch freien Speicherplatz zwischen den Zeigern FSL und STSP. Der Wert ist abhängig von der Größe des Programmtextes, der Anzahl und Art der Variablen und dem mit CLEAR reservierten Stringbereich.



Der Müll-Sammler

In der Schreibweise FRE ("") wird der für Zeichenketten zur Verfügung stehende Speicherplatz festgestellt, gleichzeitig wird der Stringbereich überarbeitet, und es werden nicht mehr benötigte Zeichenketten entfernt. Es wird mit der in das Programm eingebundenen Funktion mehr Speicherplatz während der Laufzeit des Programmes geschaffen (3.18).

```
500 PRINTPRINT("")

POKE 31470,218

100 CLEAR 1000

LIST
100 CLEAR 1000
500 PRINT
READY

RUN
1000
READY
```
3.18

```
▪FRE (Ø)
▪FRE ("")
▪MEM
```
3.19

MEM

Die Anweisung MEM ohne Parameter liefert wie FRE (0) den noch freien Speicherplatz.

BASIC-UP

Die BASIC-Erweiterung BASIC-UP gestattet die Aufnahme der vorher beschriebenen Anweisungen in das Programm. Es ist jedoch vor jeder Anweisung das Zeichen [SHIFT][J] zu geben (3.19).

Eine weitere Funktion, die sich der Zeiger der RAM-Verwaltung bedient, ist die VARPTR-Funktion. In der Form (3.21) liefert sie die Adresse der als Argument benannten Variablen, siehe auch Anhang 9.

VARPTR

Auch hier kann mit dem Token 192 für VARPTR ohne BASIC-UP gearbeitet werden. Es kann die Anweisung nach dem Verfahren wie in (3.14 - 3.18) gezeigt, implementiert werden.

Die Anweisung (3.22) schaltet BASIC-UP ab. Es werden die Zeiger der RAM-Verwaltung re-initialisiert.

```
▪VARPTR (A$)
```
3.20

```
PRINT▒VARPTR (A$)
                                    3.21
```

```
▒SYSTEM
                                    3.22
```

4 PEEK und POKE und das Integer-Format

Umrechnen natürlicher Integer-Zahlen in vorzeichenbehaftete Integer-Zahlen

Die Anweisungen PEEK (AD) und POKE AD,WERT verlangen die Adressenangabe im Integer-Format. Im Kapitel 1 ist dieses Format kurz beschrieben: Es stellt Zahlen ohne Dezimalbruch dar, die speicherintern in zwei Bytes codiert werden.

Mit zwei Bytes, entsprechend 16 Stellen binär, lassen sich Zahlen im Bereich von 0 bis 65535 darstellen. In diesem Format werden Zahlen intern vom Betriebssystem bearbeitet.

Das Zahlenformat Integer des BASIC-Interpreters wird zur Unterstützung der Arithmetik in negative und positive Zahlen aufgeteilt (4.1). Als Vorzeichen dient das höchstwertige Bit 15 (4.2). Mit den Bits 0 bis 14 können Zahlen von 0 bis 32767 dargestellt werden. Negative Zahlen werden mit gesetztem Bit 15 codiert. Sie werden als Zweierkomplement dargestellt.

Aus der Sicht des Programmierers, der mit den Anweisungen PEEK und POKE die Speicherzellen lesen oder ändern will, und der deshalb einen Zahlenbereich von 0 bis 65535 braucht, ist mit negativen Zahlen nicht viel anzufangen.

Die Zahlen 0 bis 32767 (7FFFH) werden intern in 16 Bit

```
INTEGER-ZAHLENFORMAT:
====================
-32768 -- Ø -- +32767
                                    4.1
```

```
BIT15: VORZEICHEN:
==================
Ø : POSITIV
1 : NEGATIV
                                    4.2
```

normal dargestellt und sind als Adressenangabe einsetzbar (4.3). Die nächst höhere Zahl 32768 (8000H) wird mit ihrem Wert beibehalten, bekommt aber durch das gesetzte Bit 15 ein negatives Vorzeichen, um den sonst nicht sehr sinnvollen Wert -0 zu vermeiden.

Die folgende Zahl 32769 (8001H) wird folgendermaßen konvertiert: Aus den Bits 0 bis 14 wird über das Einerkomplement das Zweierkomplement gebildet, Bit 15 wird zum Zeichen negativ' gesetzt. Bit 0 bis 14 stellen den Wert 32767 dar, mit gesetztem Bit 15 dann -32767 (4.5).

Aus diesen Erkenntnissen heraus lassen sich in (4.6) natürliche Zahlen, hexadezimale Zahlen und die Zahlen des arithmetischen Integer-Formates gegenüberstellen. Zum Umrechnen der natürlichen Zahlen in das Integer-Format dient Programm (4.7).

```
BINÄRE ZAHLEN:
==========================
    Ø  ØØØØ ØØØØ ØØØØ ØØØØ
32767  Ø111 1111 1111 1111
```
4.3

```
32768  1ØØØ ØØØØ ØØØØ ØØØØ
```
4.4

```
32769 (BIT Ø-14) ØØØ ØØØØ ØØØØ ØØØ1
1ER KOMPLEMENT   111 1111 1111 111Ø
2ER KOMPLEMENT                   +1
                 --- ---- ---- ----
                 111 1111 1111 1111
WERT             32767
VORZEICHEN       1
ZAHL             -32767
```
4.5

HEXADEZIMAL	INTEGER	NATÜRLICHE ZAHLEN
ØØØØ	Ø	Ø
ØØØ1	1	1
7FFF	32767	32767
8ØØØ	-32768	32768
8ØØ1	-32767	32769
FFFE	-2	65534
FFFF	-1	65535

4.6

```
10 K=32768
20 INPUT A
30 IF A<K PRINT A: GOTO 20
40 A=A-K
50 A=(NOT A) +1
60 A=A+K
70 PRINT -A: GOTO 20
                                4.7
```

5 Anmerkungen zu PRINT und LPRINT

PRINT, ";", ",", PRINT AT, PRINT USING, LPRINT

Die Ausgabe an Bildschirm und Drucker folgt bestimmten Regeln. Nachfolgend werden diese Funktionen eingehender erörtert. Dazu folgende Übersicht:

```
1. PRINT
2. DAS SEMIKOLON ';'
3. DAS KOMMA ',' UND TAB (X)
4. PRINT AT '@'
5. PRINT USING
6. LPRINT
```

```
PRINT 12
_12
PRINT "HALLO"
HALLO
A=3: B=5
PRINT A
_3
PRINT 3+5
_8
PRINT SQR(8)
_2.82843
A$="HALLO"
PRINT A$
HALLO
PRINT LEFT$(A$,3)
HAL
                                5.1.1
```

5.1 PRINT

Die PRINT-Anweisung gibt Daten (numerische oder Zeichenketten) an den Bildschirm aus. Nach PRINT kann eine Funktion oder ein Ausdruck genannt werden, dessen Resultat dann auf den Bildschirm ausgegeben wird (5.1.1).

In den nebenstehenden Beispielen wird das Zeichen '_' (der Unterstreichungsstrich) für ein Leerzeichen (SPACE) verwendet. Der erste wichtige Punkt: Die Ausgabe auf den Bildschirm wird vom Betriebssystem immer mit den Funktionen 'Wagenrücklauf' und 'Zeilenvorschub' abgeschlossen. Diese aus der Schreibmaschinentechnik übernommenen Steuerbefehle bewirken, daß die nächste PRINT-Anweisung die Datenausgabe am Anfang der folgenden, freien Zeile vornimmt.

Die Anweisung 'Zeilenvorschub' ist in codierter Form auch vom BASIC-Programmierer einsetzbar. Sie entspricht der Funktion 'CURSOR DOWN' '⍗' (5.1.2). Das Beispiel (5.1.3) gibt beide Anweisungen, 'Zeilenvorschub' und 'Wagenrücklauf', gemeinsam aus.

```
ZEILENVORSCHUB: KODE 10
=======================
ANWEISUNG: PRINT CHR$(10);
BEISPIEL :
?"111";CHR$(10);"222"
111
___222
                       5.1.2
```

```
WAGENRÜCKLAUF UND
ZEILENVORSCHUB: KODE 13
=======================
ANWEISUNG: PRINT CHR$(13);
BEISPIEL :
?"111";CHR$(13);"222"
111
222
                       5.1.3
```

Ein zweiter wichtiger Punkt betrifft die Ausgabe numerischer Daten (5.1.4). Hier gelten folgende Regeln:

Ein positives Vorzeichen wird durch die Ausgabe eines Leerzeichens ersetzt.

Für Zahlen kleiner als Null wird die Ausgabe der Null vor dem Dezimalpunkt unterdrückt.

Diese Einschränkungen lassen sich entweder durch kurze Wandlungsroutinen oder durch die Anweisung PRINT USING umgehen. Das Programm (5.1.5) ergänzt das positive Vorzeichen und fügt die führende Null für Zahlen <0 an. Als Unterprogramm formuliert, kann die Ausgabe numerischer Werte mit diesem Programm erfolgen.

```
10 PRINT Ø
20 PRINT -12
30 PRINT 12
40 PRINT Ø.37
50 PRINT -Ø.37
RUN
_Ø
-12
_12
_.37
-.37
                        5.1.4
```

```
10 INPUT A: A$=STR$(A)+" "
20 IF A=0 THEN 90
30 X=ASC(A$): A$=RIGHT$(A$,LEN(A$)-1):
   Y=ASC(A$)
40 IF X=32 THEN X=43
50 IF Y=46 THEN A$=CHR$(X)+"Ø"+A$
   ELSE A$=CHR$(X)+A$
90 PRINT A$: PRINT: GOTO 10
                        5.1.5
```

5.2 Das Semikolon ';'

Dieses Zeichen in einer PRINT-Anweisung bewirkt die Unterdrückung der Steuersequenz CR (Wagenrücklauf) und LF (Zeilenvorschub) am Ende der Ausgabe. Die nächste Ausgabe mit PRINT erfolgt dann in derselben Zeile unmittelbar hinter der vorherigen. (5.2.1) zeigt, daß nach der Ausgabe eines numerischen Wertes automatisch ein Leerzeichen eingefügt wird, während Zeichenketten ohne Leerzeichen aneinandergereiht werden.

Innerhalb einer PRINT-Anweisung können Funktionen, Konstante, Variablennamen und Ausdrücke in Form einer Liste

nacheinander benannt werden. Eine Trennung, wenn erforderlich, geschieht zur Ausgabe innerhalb einer Zeile durch das Semikolon (5.2.2, 5.2.3).

```
10 PRINT 2;
20 PRINT "HALLO";"X"
RUN
_2_HALLOX
                           5.2.1
```

```
PRINT 12;-.13
_12_-.13
                           5.2.2
```

```
A$="HALLO"
PRINT A$" LASER"
HALLO LASER
                           5.2.3
```

```
PRINT 1,2,3,4
1          2
3          4
                           5.3.1
```

5.3 Das Komma ',' und TAB (X)

Das Komma und die Funktion TAB (X) übernehmen in PRINT-Anweisungen Tabulator-Aufgaben. Das Komma lenkt die Ausgabe zum Bildschirm auf eine von zwei vortabulierten Positionen einer Bildschirm-Zeile. In einer Zeile mit den Spalten 0 bis 31 sind dies die Schreibstellen 0 und 16 (5.3.1).

Die TAB (X)-Funktion erzeugt innerhalb einer Ausgabezeile von 64 Zeichen (zwei Bildschirmzeilen) eine Tabulator-Position. Das Argument X von TAB (X) muß im Bereich von 0 bis 255 liegen. TAB (100) tabuliert in der Position 100-64 gleich 36 (5.3.2), während TAB (200) in der Spalte 200-(3*64) gleich 8 tabuliert. Beispiel (5.3.3) zeigt die Anwendung dieser Funktion zur rechtsbündigen Ausgabe von Zeichenketten.

```
?TAB(100);"X"
-------Ø-31-------
----X
                           5.3.2
```

```
10 A$="HAMBURG"
20 FOR I=1 TO LEN(A$)
30 PRINT TAB(15-I); LEFT$(A$,I)
40 NEXT
RUN
                    H
                   HA
                  HAM
                 HAMB
                HAMBU
               HAMBUR
              HAMBURG
                          5.3.3
```

5.4 PRINT AT '@'

Diese Anweisung ermöglicht die Ausgabe von Zahlen und Zeichenketten ab einer nach '@' definierten Position auf den Bildschirm. Als Definition kann eine Konstante, ein numerischer Ausdruck oder eine numerische Variable verwendet werden (5.4.1). Der Wert muß im Bereich von 0 bis 511 liegen, entsprechend den 512 Positionen des Bildschirmes (5.4.2).

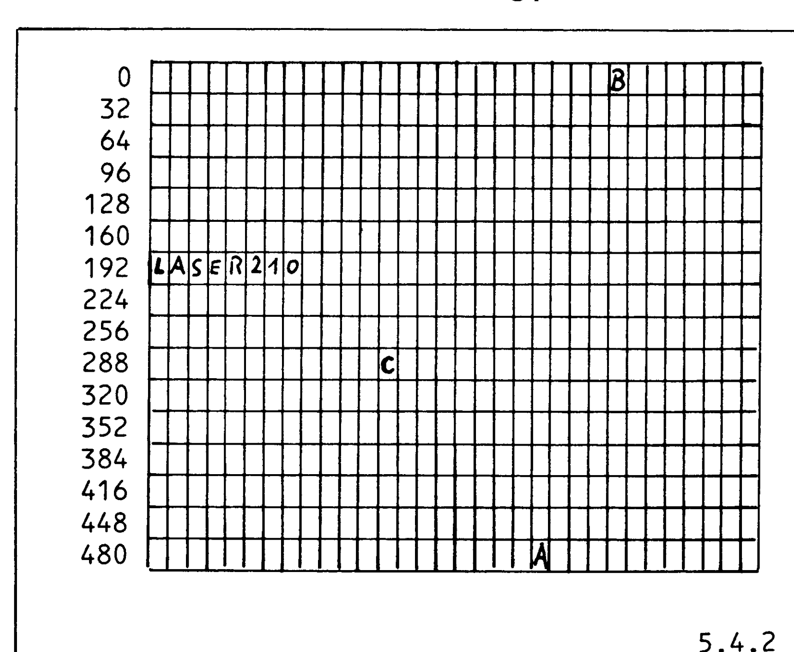
Durch die gezielte Ausgabe auf den Bildschirm lassen sich interessante Effekte erzielen. (5.4.3) gibt eine blinkende Zeile auf den Bildschirm aus. Programm (5.4.4) zeigt die Programmierung einer Eingabemaske. Hier wird die Form (5.4.5) zum Aufsetzen des Cursors auf eine mit X definierte Bildschirm-Position benutzt, um die nachfolgende INPUT-Anweisung gezielt zu plazieren. Beachten Sie in diesem Programm auch die strikte Trennung zwischen dem Bildtext in DATA-Zeilen und dem Aufbau der Maske.

```
PRINT@ 500,"A"

PRINT@ 12*2,"B"

A=150
PRINT@ A*2,"C"
                          5.4.1
```

```
10 CLS
20 PRINT@ 192,"LASER210"
30 FOR I=1 TO 100: NEXT
40 PRINT@ 192,"LASER210"
50 FOR I=1 TO 100: NEXT
60 GOTO 10
                          5.4.3
```



5.4.2

```
10 DIM Z(5,1): COLOR,1
15 '====== MASKENAUFBAU
20 CLS: RESTORE
30 FOR A=0 TO 448 STEP 64
40 READ A$: PRINT@A,A$: NEXT
50 ' ===== EINGABE
60 FOR I=0 TO 5: PRINT@140+(I*64),;:INPUT Z(I,Ø)
70 PRINT@148+(I*64),;:INPUT Z(I,1):NEXT
80 PRINT@444,"OK?":PRINT@478,"J":PRINT@476,;:INPUTJ$
90 IF ASC(J$)<>74THEN 20
91 COPY
200 '======= TEXT BILDAUFBAU
205 DATA "***** ZENSUREN-VERWALTUNG *****"
210 DATA "**FACH**   PUNKTE  ZENSUR"
220 DATA DEUTSCH,ENGLISCH,MATHE,PHYSIK,BIOLOGIE,FRANZOES.
```

5.4.4

```
***** ZENSUREN-VERWALTUNG *****

**FACH**    PUNKTE  ZENSUR

DEUTSCH      ? 23     ? 6

ENGLISCH     ? 234    ? 1

MATHE        ? 34     ? 4

PHYSIK       ? 78     ? 2
```

```
BIOLOGIE     ? 23     ? 5
                           OK?
FRANZOES.    ? 67     ? 2   ? J

                           5.4.4
```

```
100 PRINT@ X,;: INPUT A$
                            5.4.5
```

5.5 PRINT USING

PRINT USING bewirkt die formatierte Ausgabe von Zahlen. Die Anweisung USING kann in PRINT-Anweisungen einzeln oder als letzte Anweisung mit anderen formatierenden Funktionen und Steuerzeichen verwendet werden (TAB(X), ',', ';', ' @') (5.5.1). Nach USING ist stets eine Zeichenkette zu definieren, die eine Ausgabevorschrift (Maske) für die auszugebende Zahl enthält. Die Grundelemente dieser Maske sind die Zeichen '#' und '.'. Das '#' steht für eine Ziffernposition, der Punkt als Platzhalter für den Dezimalpunkt (5.5.2).

Überschreiten bei der Ausgabe die Nachkomma-Stellen die in der Maske nach dem Punkt angegebenen Stellen, so wird gerundet (5.5.3).

Werden die Stellen vor dem Dezimalpunkt der Maske von der auszugebenden Zahl überschritten, so wird die gesamte Zahl mit einem vorangestellten '%'-Zeichen ausgegeben (5.5.4).

```
USING "##.##"
                            5.5.1
```

```
PRINT USING "##.##";8.5
_8.50
                            5.5.2
```

```
PRINT USING "###";.34
__Ø
                            5.5.3
```

```
PRINT USING "##.##";315.753
%315.75
                            5.5.4
```

```
PRINT USING "##";-12
%-12
                            5.5.5
```

Für das negative Vorzeichen ist ebenfalls eine Stelle in der Maske vorzusehen. Die Zahl -12 muß mit '###' in der Maske zur Ausgabe vorbereitet sein (5.5.5).

Zur Steuerung der Vorzeichenausgabe bewirkt ein '+' in der ersten Position der Maske die Ausgabe des positiven oder negativen Vorzeichens vor der Zahl. Das '+' am Ende der Maske bewirkt die Ausgabe des Vorzeichens nach der Zahl (5.5.6).

Ein '-'-Zeichen am Ende der Maske ermöglicht die Kennzeichnung negativer Zahlen durch ein nachgestelltes Minus-Zeichen (5.5.7).

Ein doppeltes '*'-Zeichen in den ersten beiden Stellen der Maske füllt bei der Ausgabe alle nicht benutzten Positionen mit dem '*'-Zeichen (5.5.8), der sog. Schutzstern.

Das doppelte '$'-Zeichen führt zur Ausgabe eines Dollar-Zeichens unmittelbar vor der Zahl. Eine Kombination mit dem Zeichen '**' ist möglich (5.5.9).

```
PRINT USING "+###";12
 +12
PRINT USING "###+";-12
 12-
                              5.5.6
```

```
PRINT USING "###-";-12
 12-
PRINT USING "###-";12
 12
                              5.5.7
```

```
PRINT USING "**####.##";12
****12.00
                              5.5.8
```

```
PRINT USING "$$###.##";12
  $12.00
PRINT USING "$$###.##";-12
 -$12.00
PRINT USING "**$####.##";12
***$12.00
                              5.5.9
```

Ein Komma links des Dezimalpunktes trennt jeweils nach drei Stellen im ganzzahligen Teil der Zahl (5.5.10). Große Zahlen werden so besser lesbar.

Ein Ausrufungszeichen in der ersten Stelle der Maske veranlaßt die Ausgabe des ersten Zeichens einer nach der Maske angegebenen Zeichenkette. Eine Kombination mit anderen Zeichen ist möglich (5.5.11).

```
POKE 30977,8:'DBL.PREC.
A=112678.989
PRINT USING "########,.##";A
  112,678.99
                              5.5.10
```

```
A$="HAMBURG"
PRINT USING "!**###.##";A$;12.24
H***12.24
                             5.5.11
```

```
10 FORI=1TO7:READA
20 PRINT,,USING"######,.##- DM";A
30 NEXT
100 DATA.123,12.999,124375.89,-12,375,1.34E2,1.23E-2

                                      0.12 DM
                                     13.00 DM
                                124,376.00 DM
                                     12.00-DM
                                    375.00 DM
                                    134.00 DM
                                      0.01 DM
                             5.5.12
```

Alle nicht mit besonderen Funktionen versehenen Zeichen können in die Maske an den ersten oder letzten Stellen aufgenommen werden. Sie werden dann auch an diesen Positionen ausgegeben. Programm (5.5.12) demonstriert das. Grundsätzlich wird für jedes Zeichen der Maske auch eine Stelle im Bildschirm reserviert.

5.6 LPRINT

Alle mit PRINT zusammenarbeitenden Steueranweisungen arbeiten auch mit LPRINT: Ausgabe zum Drucker, zusammen. Jedoch sind einige besondere Eigenschaften zu beachten.

Das Komma tabuliert in einer Zeile, die 128 Zeichen umfaßt. In jeder 16. Spalte ist eine Position vortabuliert. Wird ein Drucker mit 80 Zeichen/Zeile verwendet, so werden in der ersten Zeile 5 vortabulierte Positionen angesteuert. Der Abstand beträgt auch hier jeweils 16 Zeichen.

Werden mit einer LPRINT-Anweisung mehr als 5 Tabulator-Positionen angesteuert, so werden in der zweiten Zeile noch drei weitere Tabulatoren gesetzt (5.6.1).

PRINT USING arbeitet in der Form LPRINT USING wie auf den Bildschirm.

Ebenfalls wird das Steuerzeichen ';' mit LPRINT zusammenarbeiten wie von der Bildschirmausgabe gewohnt.

Die Funktion TAB (X) tabuliert von X=0 bis X=63 in einer Zeile. TAB (64) tabuliert wieder in Spalte 0. Mit LPRINT TAB (X) kann man also nur in die Spalten 0 bis 63 tabulieren.

```
LPRINT 1,2,3,4,5,6,7,8,9,10,11,12,13,14
..............................................
 1               2               3               4       5
 6               7               8
 9               10              11              12      13
14
                                                      5.6.1
```

6 Hilfen für den Programmierer

STOP, CONT, TRACE ON, TRACE OFF, LIST, DELETE, AUTO

Voraussetzung für eine effiziente Programm-Editierung und Fehlersuche ist eine blockförmige Programmstruktur. D.h. der Programmierer sollte immer wiederkehrenden Strukturen in seinem Programm feste Bereiche der Zeilennummern zuordnen. Schon beim Listen des Programmes auf den Bildschirm kann so gezielt auf einzelne Abschnitte zugegriffen werden. Der nachfolgende Rahmen für ein BASIC-Programm fußt zum Teil auf programmtechnischen Überlegungen. So ist zum Beispiel das Auffinden von Unterprogrammen für den Interpreter wesentlich schneller, wenn sie am Anfang des BASIC-Programmes stehen.

```
10      GOTO 20000: SPRUNG ZUM HAUPTPROGRAMM
100     UNTERPROGRAMME, Z.B. RECHNEN, STRINGMANIPULATIONEN
500     SCHLEIFEN MIT INKEY$
1000    INPUT VON DER FLOPPY
2000    PRINT AUF DIE FLOPPY
3000    DRUCKER:HAUPTUNTERPROGRAMME
-       FORMATIERUNGS-UNTERPROGRAMME
5000    AUSGABE AUF ANDERE PERIPHERIE ( KASSETTE)
-       FORMATIERUNGSANWEISUNGEN FÜR ANDERE PERIPHERIE
8000    DATA - READ - ROUTINEN
10000   DATA-ZEILEN, GESAMMELT
20000   HAUPTPROGRAMM *********
-       INITIALISIERUNG DER AM HÄUFIGSTEN GEBRAUCHTEN
        VARIABLEN MIT DIM
-       MENUE ZUR HAUPT-ABLAUFSTEUERUNG
-       BEFEHLSEBENE MIT ON...GOTO 30000,40000,....
```

6.Ø

In einem so aufgebauten Programm läßt sich mit dem LIST-Kommando zum Überarbeiten des Programmes schnell auf ein Teilstück zugreifen.

LIST

Der Vollständigkeit halber sei das LIST-Kommando hier kurz angesprochen: LIST (CR) rollt das gesamte Programm über den Bildschirm. Schnell-Leser können die Ausgabe an der gewünsch-

```
LIST
LIST 1Ø
LIST 1ØØ-2ØØ
LIST -1ØØØ
LIST 1ØØ-
```

6.1

```
10 INPUT "DEZ.ZAHL Ø-255";D
20 FOR I=7 TO Ø STEP -1
30 PRINT SGN (D AND 2↑I);
40 NEXT I: PRINT
50 LIST 20-40
RUN
DEZ.ZAHL Ø-255?  127
Ø  1  1  1  1  1  1  1
20 FOR I=7 TO Ø STEP -1
30 PRINT SGN (D AND 2↑I);
40 NEXT I: PRINT
```

6.2

ten Stelle durch Betätigen der Leertaste (SPACE) anhalten und mit nochmaligem Drücken fortsetzen. (6.1) zeigt die möglichen Parameter des LIST-Kommandos. Der '–' steht für 'von Anfang bis ...' oder 'bis Ende'. Übrigens kann LIST als letzte Zeile im Programm stehen. Ein Probelauf wird dann immer mit dem Auslisten der gerade bearbeiteten Routine abgeschlossen (6.2). (Anm: Diese kleine, aber wirkungsvolle Routine wandelt eine Dezimalzahl in eine Binärzahl!)

BASIC-UP: DELETE

Eine weitere Hilfestellung wird wieder von der BASIC-UP-Erweiterung bereitgestellt: die DELETE-Anweisung. Sie wird mit Parameter gegeben und löscht gruppenweise die Anweisungszeilen im Programmlisting wie im Parameterteil angegeben. Es ist möglich, mit DELETE # eine Zeile zu löschen, jedoch wird man hier lieber die Zeilennummer, gefolgt von RETURN, geben. Weiter kann mit DELETE # - # ein Block im Programm von Zeile X bis Zeile Y gelöscht werden. DELETE –Y löscht vom Programmanfang bis zur angegebenen Zeilennummer.

Als Editieranweisung sollte DELETE nicht im Programm benutzt werden. Eine entsprechende Fehlermeldung wird dann ausgegeben.

```
█DELETE 1Ø
█DELETE 2ØØ - 3ØØ
█DELETE - 1ØØ
                    6.3
```

AUTO - Automatische Zeilennummern-Vorgabe

Die Anweisung AUTO X,Y gibt im Editier-Modus automatisch die jeweils nächste Zeilennummer vor, wobei X die Zeilennummer ist, ab der die Vorgabe beginnt. Y ist die Schrittweite. AUTO 10,10 erzeugt Zeilennummern ab 10 in 10er-Abstand, also 10, 20, 30 usw.

Wird AUTO ohne Parameter eingegeben, so wird automatisch auf 10,10 geschaltet. AUTO 350,50 numeriert ab 350 mit Schrittweite 50, AUTO ,5 numeriert ab 10 mit Schrittweite 5 und AUTO 130 entspricht 130,10. Das Betriebssystem hält alle Routinen dieser Funktion vor. Das Wort 'AUTO' wird nicht erkannt, jedoch kann die Funktion mit drei POKE-Anweisungen

eingeschaltet werden (6.4). Mit CTRL-BREAK wird die automatische Numerierung abgebrochen. Wird die letzte POKE-Anweisung (6.4) erneut gegeben, so wird der AUTO-Betrieb mit der nächsten, folgenden Nummer wieder aufgenommen.

```
AUTO EINSCHALTEN:
=================
(Methode 1)

POKE 30946,10
  (STARTNUMMER LSB)
POKE 30947,Ø
  (STARTNUMMER MSB)

POKE 30948,1Ø
  (SCHRITTWEITE LSB)
POKE 30949,Ø
  (SCHRITTWEITE MSB)


POKE 30945,255
  (START, RESTART)
                        6.4.1
```

```
AUTO EINSCHALTEN
================
(Methode 2)

10 PRINT 1Ø,1Ø: 'ERSTE ZEILE!

(VON DER TASTATUR:)
POKE 31469,183: 'TOKEN AUTO

(START AUTO-FUNKTION:)
RUN

(RESTART DER AUTO-FUNKTION
NACH BREAK:)

POKE 30945,255
                        6.4.2
```

BASIC-UP - AUTO X,Y

Der AUTO-Modus kann mit der BASIC-Erweiterung BASIC-UP eingeschaltet (6.5) und ebenfalls mit CTRL-BREAK unterbrochen werden.

```
█AUTO
█AUTO 35Ø, 5Ø
█AUTO ,5
█AUTO 13Ø
                        6.5
```

TRACE ON - TRACE OFF,
Schrittweises Abarbeiten des BASIC-Programmes

Eine weitere Funktion des Betriebssystemes, die ebenfalls nur mit POKE-Anweisungen ein- und ausgeschaltet werden kann, ist die schrittweise Abarbeitung des Programmes oder eines Programmteiles mit Ausgabe aller bearbeiteten Zeilennummern auf den Bildschirm.

Es lassen sich mit dieser Funktion besonders Verzweigungsanweisungen wie IF...THEN...ELSE und GOTO, GOSUB, ON...GOTO... und ON...GOSUB... testen. Die Anweisungen zum Einschalten dieser Betriebsart ('TRACE ON') und zum Abschalten ('TRACE OFF') zeigt (6.6). Das Beispiel (6.7) wird das Programm zum Umrechnen von dezimalen Zahlen in binäre (6.2) im TRACE-Betrieb ausgeben. Deutlich wird hier die Arbeitsweise: Eingeben von der Tastatur und Ausgeben auf den Bildschirm werden normal angezeigt, während in spitzen Klammern nacheinander die bearbeiteten Zeilennummern auf den Bildschirm gegeben werden. Eine genaue Programmanalyse ist so möglich.

```
POKE 31003,63: 'TRACE ON

POKE 31003,0 : 'TRACE OFF
                           6.6
```

```
RUN
DEZ.ZAHL 0-255? 127
 0  1  1  1  1  1  1  1
READY

POKE 31003,63
READY    •
RUN
<10>DEZ.ZAHL 0-255? 127
<20><30> 0 <40><30> 1 <40><30> 1
  <40><30> 1 <40><30> 1 <40><30>
 1 <40><30> 1 <40><30> 1 <40>
READY
COPY
                           6.7
```

STOP und CONT und CTRL/BREAK

Die regulären BASIC-Anweisungen STOP und CONT sind

ebenfalls zum Austesten der vom Anwender geschriebenen BASIC-Programme vorgesehen.

Die Taste CTRL/BREAK gleicht in ihrer Wirkung der Anweisung 'STOP'. Beide unterbrechen den Programmablauf, STOP an definierter Stelle im Programm und CTRL/BREAK bei Betätigung dieser Tastenkombination. CONT steht für CONtinue und setzt das Programm mit der nächsten zu bearbeitenden Anweisung fort. - Wie nun können diese Anweisungen zum Auffinden von Fehlern eingesetzt werden? Dies sei an einem Beispiel erklärt:

In einem einigermaßen strukturiert aufgebauten Programm (siehe Anfang dieses Kapitels!) wird in einem Programmteil ein Fehler vermutet. Vor dem Beginn und am Ende des Programmteiles wird jeweils eine 'STOP'-Anweisung eingefügt. Mit Erreichen der ersten STOP-Anweisung wird das Programm mit der Meldung 'STOP in XX' abgebrochen, wobei XX die Zeilennummer ist, in der die STOP-Anweisung ausgeführt wurde. Es besteht nun für den Programmierer die Möglichkeit, Daten für den Programmteil neu zu definieren, das Programm mit der Anweisung 'CONT' fortzusetzen und mit dem zweiten STOP das Ergebnis der Datenverarbeitung von der Tastatur her abzufragen.

Eine schrittweise Kontrolle, nicht des Programmlaufes wie bei TRACE ON', sondern der Bearbeitung aller Daten, ist so möglich. Beispiel (6.8) zeigt die Bildschirmkopie einer solchen Arbeit. Ein Programm, das mit der CTRL/BREAK-Taste abgebrochen wurde, läßt sich ebenfalls mit CONT wieder aufnehmen. Für beide Arten des Abbruches muß einschränkend gesagt werden, daß nach Änderungen im Programmlisting nicht wieder mit CONT gestartet werden kann. Ein neuer Programmstart mit RUN ist dann unter Verlust aller bisher erarbeiteten Daten notwendig.

Soll ein mit STOP abgebrochenes Programm an anderer Stelle fortgesetzt werden, so kann von der Tastatur mit GOTO XX ohne Datenverlust der Programmlauf an der mit XX bezeichneten Stelle fortgesetzt werden.

Zum Beispiel (7.8): In einer Schleife werden vier Feldvariablen mit dem Quadrat der Schleifenvariablen belegt. In einer mit STEP -1 rückwärts laufenden Schleife werden die Ergebnisse

ausgegeben, jedoch nicht vier, sondern fünf. Die Zeile 35 STOP wird zum Test eingegeben. Der Probelauf zeigt dann, daß der Schleifenzähler nach Beendigung der Zeile um 1 größer ist, als der mit TO definierte Wert. Die Ausgabeschleife wird dann also 5 Ausgaben auf den Bildschirm bringen.

```
10 FOR I=1 TO 4
20 A(I)=I↑2
30 NEXT I
40 FOR I=I TO 1 STEP -1
50 PRINT A(I);
60 NEXT
RUN <CR>

0  16  9  4  1
READY

35 STOP <CR>
RUN <CR>

BREAK IN 35
READY

PRINT I,A(I) <CR>
5     0
```

6.8

7 Textgrafik, Funktion und Anwendung

2000 Zeichen Auflösung, Viertelgrafik, ein Viertel lernt laufen

Der Bildschirm des LASER kann im Textbetrieb (MODE (0)) 512 Zeichen darstellen: 16 Zeilen*32 Zeichen. Jede Speicherzelle des Video-RAM läßt sich mit der Anweisung POKE beschreiben und mit PEEK lesen. In BASIC-Programmen läßt sich eine Speicherzelle mit der Basisadresse 28672 und den Koordinaten X (Position in der Zeile) und Y (Zeilennummer) beschreiben, wobei X die Werte 0 bis 31 und Y die Werte 0 bis 15 annehmen kann. Eine Zeichenposition läßt sich also mit 28672 + (Y*32) + X beschreiben. In der im Anhang dargestellten Aufteilung des Bildschirmes liegt die markierte Zeichenposition in der Zeile 8 und der Spalte 7. Die zugehörige RAM-Speicheradresse ist dann:

$$28672 + (8*32) + 7 = 28935$$

Mit POKE adresse,code ist so jedes Zeichen gezielt auf den Bildschirm zu bringen. (Codes: Siehe Anhang 13.)

Das Beispiel (7.3) bringt in die 12. Zeile, 20. Spalte ein 'A'. Beachten Sie, daß die Spalten- und Zeilennumerierung jeweils mit '0' beginnt! Beachten Sie auch, daß der Code für den Bildschirm nicht mit dem ASCII-Code identisch ist!

2000 Zeichen Auflösung bei Grafik!

Der Grafik-Zeichensatz umfaßt 16 Zeichen der Viertel-Grafik. Für die 8 Farben des Video-Controllers ist jeweils ein Zeichensatz vorhanden.

Mit der Viertel-Grafik wird die Auflösung im Textbetrieb von 512 auf 2048 Punkte gesteigert!

Um die 8*16 Zeichen nun gezielt einzusetzen, soll nachfolgend der Aufbau und die Organisation dargestellt werden.

```
VIDEO-RAM:
----------
28672 - 29183
7ØØØH - 71FFH
                                  7.1
```

```
ZEICHENADRESSIERUNG:
--------------------
28672 + (Y*32) + X
                                  7.2
```

```
10 X=20: Y=12
20 BR=28672
30 POKE BR+(Y*32)+X,1
                                  7.3
```

Jede Bildschirmposition ist in vier Rechtecke unterteilt. Die Kombination gesetzter und gelöschter Viertel ermöglicht die Darstellung von 16 Zeichen. Wird der Bildschirmcode eines Grafikzeichens binär dargestellt, so ist in den vier niederwertigen Bits das Zeichen codiert, die Bits 4-6 beschreiben die Farbe und Bit 7 zeigt gesetzt die Grafik an.

Wenden wir uns jetzt den Bits 0-3 zu: Mit diesen vier Bits lassen sich alle 16 Grafikzeichen codieren. Bit 0 beschreibt das rechte untere Viertel, Bit 1 das linke untere. Entsprechend wird Bit 2 rechts oben und Bit 3 links oben anzeigen. Wird im Grafik-

zeichen ein Viertel gesetzt, so wird die entsprechende Bitposition auf 1 gesetzt.

Das Grafikzeichen wird also durch die Binärzahl 0100 dargestellt.

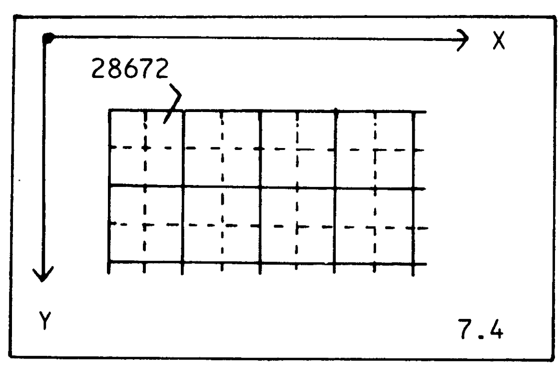
BASIC läßt keine Binärzahlen zu. Um zum Erstellen von Grafiken in BASIC den notwendige Code dezimal zu errechnen, ist jedem Viertel ein 'Wert' zugeordnet (7.7). Der Zeichencode errechnet sich als Addition der Werte der gesetzten Viertel. Beispiel:

$1 + 8 = 9$

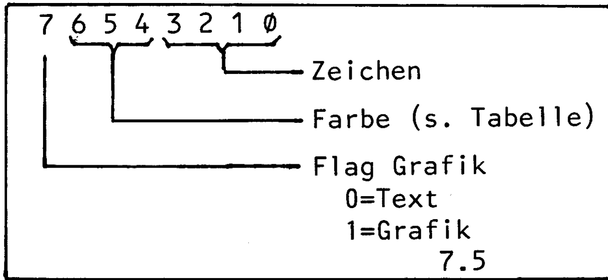
$1 + 2 = 3$

$1 + 2 + 8 = 11$

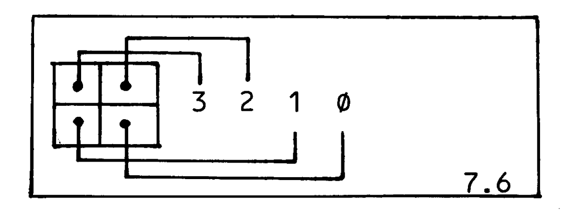
Vergleichen Sie mit der nebenstehenden Tabelle! Mit dieser dezimalen Zahl haben wir aber noch nicht die Codezahl des Grafikzeichens. Addiert werden muß noch die Farbwertigkeit zum Zeichencode sowie der Wert für Grafik = 128!



7.4



7.5



7.6

Beispiel: Berechnung des Codes für das Zeichen  mit der Farbe 'cyan':

Zeichencode	= 1 + 2 + 8	=	11
Farbe	= Cyan	=	80
Grafikwert	=		128
Mit POKE zu setzender Code		=	219

Verwenden Sie das Beispiel-Programm 7.3 zur Kontrolle!

8	4
2	1

7.7

Nr.	Grafik	Code
0		0000
1		0001
2		0010
3		0011
4		0100
5		0101
6		0110
7		0111
8		1000
9		1001
10		1010
11		1011
12		1100
13		1101
14		1110
15		1111

7.8

NR.	FARBE	KODE	WERT
Ø	GRÜN	ØØØ	Ø
1	GELB	ØØ1	16
2	BLAU	Ø1Ø	32
3	ROT	Ø11	48
4	BRAUN	1ØØ	64
5	CYAN	1Ø1	8Ø
6	MAGENTA	11Ø	96
7	ORANGE	111	112

7.9

```
10 GR=128: BR=28672
20 FORC=ØTO127 STEP16
30 FORZ=1 TO15
40 CH=GR+C+Z
50 POKE BR,CH
60 BR=BR+1
70 NEXT Z,C
```

7.10

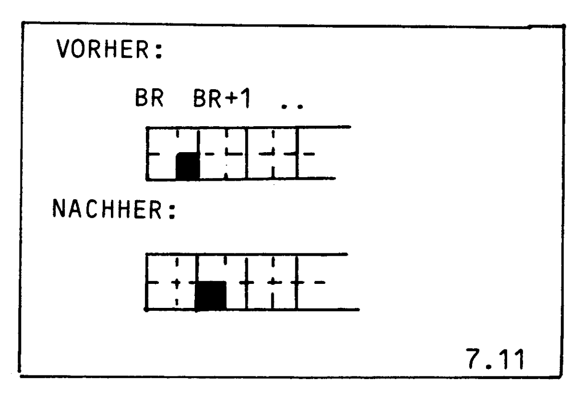
Aus dem bisher Erfahrenen kann nun ein Demonstrationsprogramm konstruiert werden, das die Grafikzeichen in allen 8 Farben berechnet und auf den Bildschirm bringt (7.10).

Um nun auf dem Bildschirm grafisch zu arbeiten, müssen die Zeichen manipuliert werden können. Dazu muß jedes Viertel

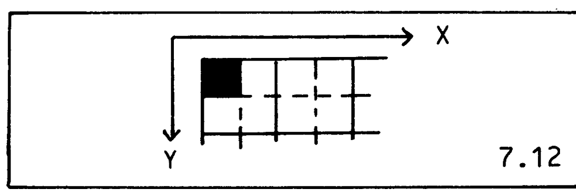
des Grafikzeichens gelesen, gesetzt und gelöscht werden können. Beispiel:

Ein gesetztes Viertel unter der Video-RAM-Adresse BR soll um ein Viertel-Quadrat nach rechts versetzt werden (7.11). Folgende Schritte sind notwendig:

1. Video-RAM unter BR lesen.
2. Feststellen, welches Viertel gesetzt ist.
3. Ist das linke Viertel oben oder unten gesetzt, dann ...
... linkes Viertel löschen,
... rechtes Viertel setzen,
... und unter BR in das RAM eintragen.
4. ist das rechte Viertel oben oder unten gesetzt, dann ...
... BR um eins erhöhen.
... rechtes Viertel löschen,
... linkes Viertel setzen,
... und unter BR eintragen.



7.11



7.12

```
10 BR=28672
20 POKE BR,200
30 IF BR<29184 THEN
    GOSUB 100: GOTO 30
40 END
100 FC=INT(PEEK(BR)/16)*16
```

```
110 ZC=PEEK(BR)-FC
120 POKE BR,FC
130 IF ZC=8
     THEN ZC=ZC AND 7 OR 4
     ELSE ZC=ZC AND 11 OR 8
     : BR=BR+1
140 POKE BR, ZC+FC
150 RETURN
                              7.13
```

```
A = PEEK (BR)
                              7.14
```

Ein "Viertel" lernt Laufen!

Das folgende Programm bewegt ein Viertel-Quadrat über den Bildschirm! (7.13)

Zeile 10 legt den Anfang des Bildschirm-RAM fest, Zeile 20 schreibt das erste Grafikzeichen ein. Die folgende Zeile stellt das Hauptprogramm dar, das solange das Unterprogramm 100 aufruft, bis das Ende des Video-RAM erreicht ist. Die Zeilen 100 und 110 lesen unter BR das letzte Zeichen und stellen Grafik- und Farbcode in FC und den Zeichencode in ZC ein. Zeile 120 löscht dann das letzte Zeichen. Zeile 130 errechnet den neuen Zeichencode als Funktion des alten Codes und erhöht, wenn notwendig, die Video-RAM-Adresse BR. Die Zeile 140 schreibt das neue Zeichen auf den Bildschirm, in 150 erfolgt die Rückkehr zum Hauptprogramm. - Lassen Sie uns nun Zeile 130 noch etwas genauer ansehen!

Da wir in Zeile 20 das erste Grafikzeichen mit dem Code 200 initialisiert haben, arbeiten wir mit der Farbe 'Braun' und der Viertel-Wertigkeit = 8. In ZC haben wir also im ersten Durchlauf eine 8 zu erwarten.

Wenn ZC = 8, DANN ist das Zeichen =
SONST ist es ein mit Wert = 4.

Wenn es eine 8 ist, dann wird das Zeichen mit dem AND-Operand 7 verknüpft:

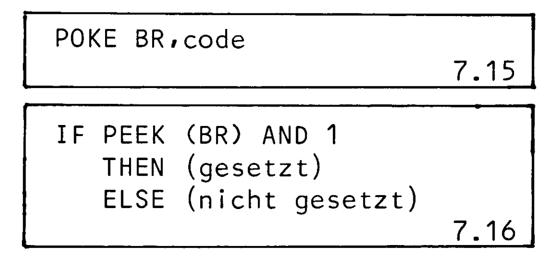
	8	1000 .....
AND	7	0111 .....
Ergebnis	0	0000 .....

Das Zeichen ist gelöscht!

	0	0000 .....
OR	4	0100 .....
Ergebnis	4	0100 .....

Das rechte obere Viertel ist gesetzt!

Vollziehen Sie die Verknüpfung nun für den ELSE-Teil der Anweisung nach! Dort muß aus einer erkannten 4 wieder eine 8 werden. Zum Wechsel von 4 nach 8 muß außerdem die Video-RAM-Adresse erhöht werden.



Nachfolgend finden Sie eine Zusammenfassung der BASIC-Anweisungen zum Lesen, Schreiben, Testen, Setzen und Löschen der Grafikzeichen und Grafikviertel:

* Lesen einer Bildschirmzelle (7.14).
* Schreiben einer Bildschirmzelle (7.15).
* Test, ob ein Viertel gesetzt oder gelöscht ist (7.16).

AND 1 testet Viertel mit Wert 1
AND 2 testet Viertel mit Wert 2
AND 4 testet Viertel mit Wert 4
AND 8 testet Viertel mit Wert 8

* Ein Viertel wird gelöscht (7.17).

AND 210 löscht Viertel mit Wert 1
AND 195 löscht Viertel mit Wert 2
AND 165 löscht Viertel mit Wert 4
AND 105 löscht Viertel mit Wert 8

(Farbcode und Grafikwert werden nicht verändert!)

* Ein Viertel wird gesetzt (7.18).

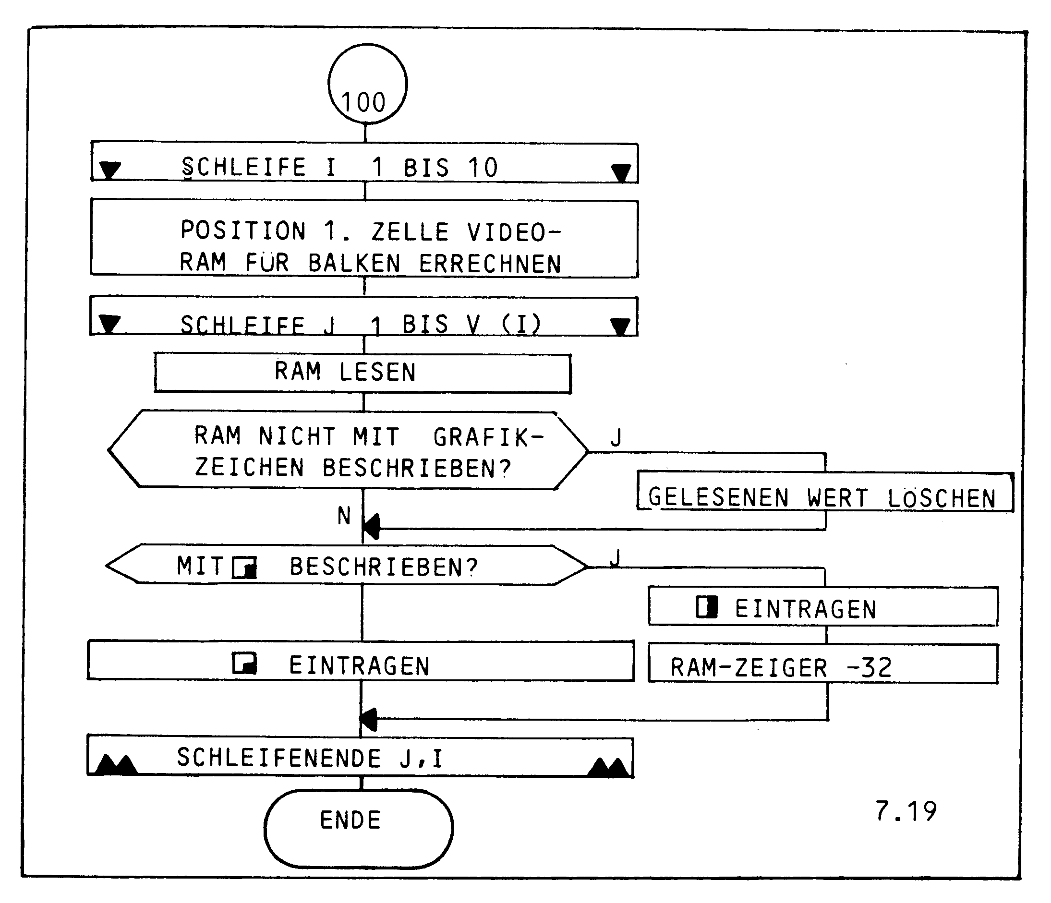
OR 1 setzt Viertel mit Wert 1
OR 2 setzt Viertel mit Wert 2
OR 4 setzt Viertel mit Wert 4
OR 8 setzt Viertel mit Wert 8

Natürlich lassen sich mit entsprechenden Operanden auch mehrere Viertel gleichzeitig setzen, testen und löschen.

```
POKE BR, PEEK(BR) AND 210
                     7.17
```

```
POKE BR, PEEK(BR) OR 1
                     7.18
```

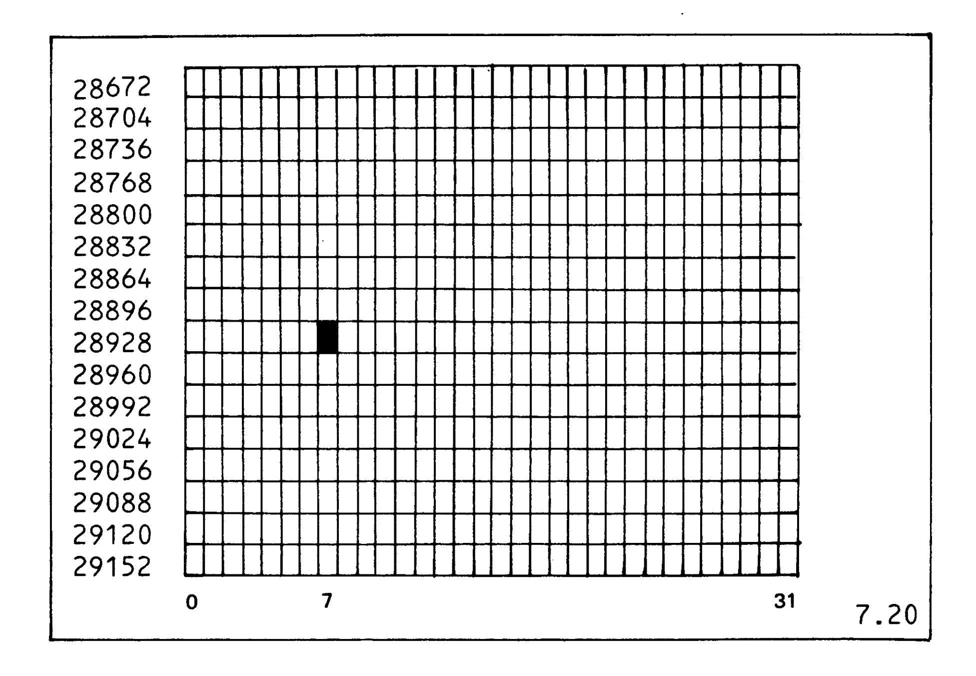
Das nachfolgende Programm erzeugt ein Balkendiagramm zur grafischen Darstellung von 10 Zahlenwerten in V(I) mit Farben, die im Feld F(I) abgelegt sind. Versuchen Sie nun, die Funktion dieses Programmes anhand des Flußdiagrammes nachzuvollziehen.



7.19

```
10 CLS
20 FOR I=1 TO 10: READ V(I),F(I): NEXTI
30 DATA 2,128,12,144,14,160,15,176,21,192
40 DATA 23,208,8,224,14,240,23,128,22,144
```

```
100 FOR I=1 TO 1Ø
110   Y=15: X=4+(I*2): GOSUB 210
120   FOR J=1 TO V(I)
130     A=PEEK(PR): IF A=32 OR A=96 THEN A=Ø
140     IF A AND F(I)+1 THEN POKE PR, A OR F(I)+5
        : GOSUB 2ØØ: GOTO 16Ø
150     POKE PR, A OR F(I)+1
      NEXT J
170 NEXT I
180 END
200 Y=Y-1
210 PR=28672+(Y*32)+X: RETURN
```



7.20

8 Mehr Grafik und Ton

Das Zeichen "←", INVERS, Cursor-Steuerung, COLOR mit dem LASER 110, MODE und Hintergrundfarbe, BUZZER ein-aus

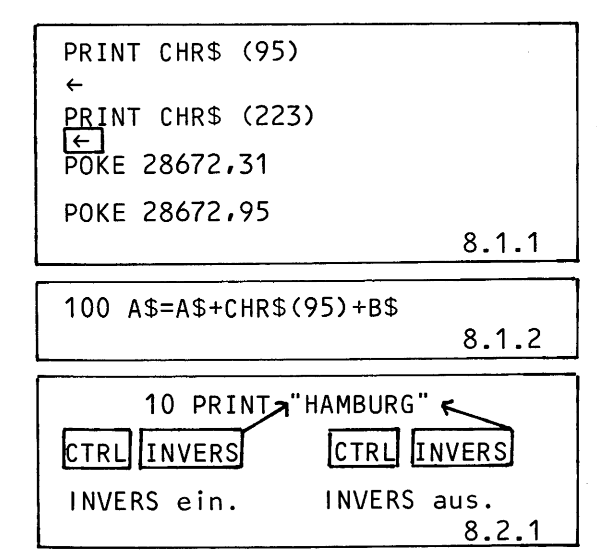
8.1 Das Zeichen "←"

Im Zeichensatz des LASER ist ein Zeichen vorhanden, welches nicht über die Tastatur erzeugt werden kann: Es ist "←". Es kann jedoch mit den Anweisungen wie in (8.1.1) an Zeichenketten an- oder eingekettet werden oder, wie in (8.1.2), direkt auf den Bildschirm gebracht werden.

8.2 INVERS

Zeichenketten können als Konstanten INVERS auf den Bildschirm gebracht werden (8.2.1).

Die Tastenkombination [CTRL] [INVERS] setzt oder löscht ein Bit im Flagregister 7838H des Betriebssystems. Die Ausgabe von Variablen und Konstanten auf den Bildschirm kann so durch Setzen oder Löschen des INVERS-Flag normal oder INVERS



```
PRINT CHR$ (95)
←
PRINT CHR$ (223)
[←]
POKE 28672,31

POKE 28672,95
```
8.1.1

```
100 A$=A$+CHR$(95)+B$
```
8.1.2

```
10 PRINT "HAMBURG"
```
[CTRL] [INVERS] INVERS ein. [CTRL] [INVERS] INVERS aus.

8.2.1

```
POKE 30776,PEEK(30776) OR 2
                          8.2.2
```

```
POKE 30776,PEEK(30776) AND 253
                          8.2.3
```

```
10 A=123.375:F=30776
20 POKE F,PEEK(F) OR 2
30 PRINT A,F
40 POKE F,PEEK(F) AND 253
50 PRINT A,F
RUN
123.375       30776
123.375       30776
                          8.2.4
```

erfolgen. (8.2.2) setzt das Flag, alle im Programm folgenden Ausgaben werden INVERS abgebildet. (8.2.3) löscht das Flag, die folgenden Ausgaben zum Bildschirm werden normal gebracht. Das kleine Programm (8.2.4) zeigt die Anwendung.

8.3 Cursor-Steuerung

Alle Cursor-Steuerbefehle, die beim Editieren der Programme von der Tastatur gegeben werden, lassen sich auch in das Programm aufnehmen. Menue-Steuerungen, Eingabe-Masken, -Absicherungen und Grafiken lassen sich so optimal erstellen (8.3.1). Die Ausführung dieser Funktionen im Programm wird mit PRINT CHR$ (Code) programmiert, oder der Code wird mit dem String-Verkettungsoperator "+" an oder in einen String gekettet. Das Beispiel (8.3.2) sichert eine Eingabe ab.

Beispiel (8.3.3) ermöglicht zur Eingabe verschiedener Werte die Vorgabe eines häufig benutzten Wertes (Default-Wert). Dieser Wert kann dann entweder mit RETURN übernommen werden, oder er wird überschrieben.

```
Cursor links         8
Cursor rechts        9
Cursor aufwärts     27
Cursor abwärts      10
Cursor 'Home'       28

'Clear Screen'      31
CR (RETURN)         13
Insert              21
Rubout             127
                          8.3.1
```

Die Cursor-Steuercodes lassen sich mit INKEY$ von der Tastatur übernehmen und weiterverarbeiten.

```
100 INPUT "WELCHES FACH";K
110 IF K<1 OR K>7 THEN PRINT CHR$(27);CHR$(27): GOTO 100
                                                  8.3.2
```

```
10 PRINT "IHRE WAHL__2";CHR$(8);CHR$(8); CHR$(8)
20 INPUT I
RUN
IHRE WAHL? 2
                                  8.3.3
```

8.4. COLOR ... mit dem LASER 110

Obwohl der LASER 110 keine Farbe auf dem Bildschirm erzeugen kann, kennt er die Anweisung COLOR. Sie kann sinnvoll eingesetzt werden.

Die Tastenkombination CTRL/N erzeugt das Wort 'COLOR' auf dem Bildschirm. Wird die Anweisung mit dem Parameter 1, wie in (8.4.1) gegeben, so verbessert sich der Kontrast des Bildes auf dem Monitor oder TV-Empfänger.

Weiterhin lassen sich Grafiken in verschiedenen Grauwerten darstellen. Eine Tabelle der Grauwerte erzeugt das Programm (8.4.2).

Da der LASER 110 mit dem gleichen Betriebssystem arbeitet, wie es der Color-Computer LASER 210 besitzt, kann der PAL-Farbcoder ohne weiteres nachgerüstet werden (Anhang 8). Die BASIC-Anweisung COLOR wird dann einwandfrei arbeiten und wie bei den Color-Computern LASER 210, 310 und VZ200 die angewiesene Farbe erzeugen.

```
COLOR,1 <RETURN>
                        8.4.1
```

```
10 FOR I=1 TO 8
20 COLOR I
30 PRINT I;"      ": 'SHIFT J
40 NEXT I
                        8.4.2
```

8.5 MODE und Hintergrundfarbe

Der Video-Controller des LASER läßt zwei Betriebsarten zu:

1. Text und Viertelgrafik
2. Hochauflösende Grafik

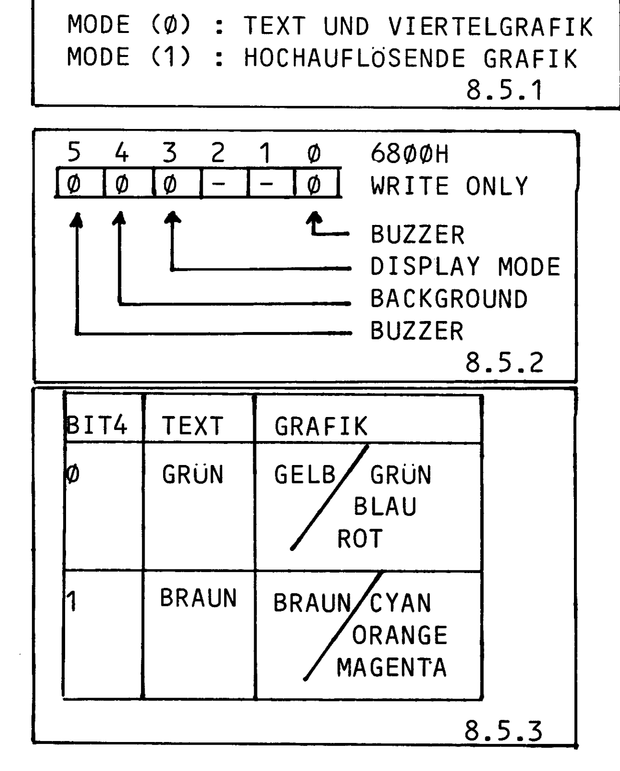
Die Umschaltung erfolgt in BASIC mit der Anweisung MODE (x). Nach dem Einschalten ist die Text-Betriebsart initialisiert. Mit MODE (1) wird auf hochauflösende Grafik geschaltet. MODE (0) schaltet zurück auf den Textbetrieb (8.5.1).

Die Steuerung der Umschaltung erfolgt über das 'Output Latch' (8.5.2). Bit 3 schaltet auf Textbetrieb, eine 1' schaltet die hochauflösende Grafik ein.

Bit 4 steuert entsprechend die Hintergrundfarbe und selektiert im 'hochauflösenden Grafik-Betrieb' je drei Vordergrundfarben (siehe Tabelle 8.5.3).

In BASIC kann die Hintergrundfarbe wie in (8.5.4) direkt oder aus dem Programm heraus bestimmt werden, ohne daß die Vordergrundfarbe genannt wird.

Das Steuern der Betriebsart und der Hintergrundfarbe mit der POKE-Anweisung erübrigt sich, da mit den Anweisungen MODE (x) und COLOR leistungsfähige Möglichkeiten für den BASIC-Programmierer vorhanden sind.



BIT4	TEXT	GRAFIK
Ø	GRÜN	GELB / GRÜN BLAU ROT
1	BRAUN	BRAUN / CYAN ORANGE MAGENTA

8.5.3

```
COLOR,Ø : GRÜN
COLOR,1 : BRAUN
                        8.5.4
```

8.6 BUZZER ein-aus

Bit 0 und Bit 5 des in 8.5. beschriebenen 'Output Latch' steuern den Buzzer. Auch hier wird normalerweise die Ansteuerung mit der BASIC-Anweisung SOUND vorgenommen werden. Jedoch unterhält das Betriebssystem unter der Adresse 783BH/ 30779 eine Kopie des Output Latch. Über dieses Register läßt sich der Buzzer ein- und ausschalten (8.6.1).

```
POKE 30779,Ø : BUZZER 'AUS'
POKE 30779,1 : BUZZER 'EIN'
                        8.6.1
```

9 Peripherie-Steuerung mit INP und OUT

USER-Port: Steuerung mit INP und OUT
Schaltungen und Beispiele in BASIC

Der LASER-Computer (LASER 110, 210, 310,VZ200) verfügt über ein System der Interface-Steuerung, das es ihm ermöglicht, Daten und Steuerbefehle mit externen Geräten auszutauschen.

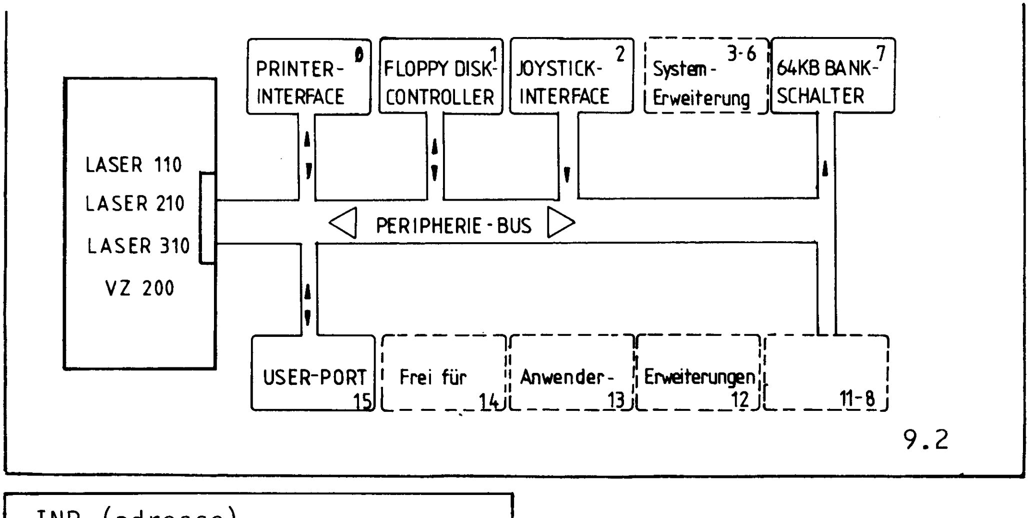
Dieser Datentausch erfolgt unter insgesamt 256 Adressen, von denen je 16 einem vorhandenen oder möglichen Gerät zugeordnet sind.

Von diesen 16 Ein- und Ausgabekanälen sind die ersten acht für Systemfunktionen reserviert. Das Betriebssystem steuert über

Ein-Ausgabekanäle der LASER-Computer

Adr. (10)	Adr. (16)	Gerät
0 - 15	00H - 0FH	Printerinterface (Drucker)
16 - 31	10H - 1FH	Floppy Disk Controller
32 - 47	20H - 2FH	Joystick Interface
48 - 111	30H - 6FH	Reserviert für zukünftige Erweiterungen des Systems
112 - 127	70H - 7FH	64 KB-RAM Memory Bank Schalter
128 - 239	80H - EFH	Frei für Anwender-Erweiterungen
240 - 255	F0H - FFH	User-Port-Modul

9.1



9.2

```
INP (adresse)
A= INP (X)
```

9.3

```
OUT adresse,datum
OUT 254,130
```

9.4

diese Kanäle Drucker und Floppy-Disk-Controller, liest die Joysticks und schaltet die RAM-Speicherbänke der RAM-Erweiterung (9.1 und 9.2).

Die anderen acht Kanäle sind für Anwenderfunktionen vorgesehen. Hier ist inzwischen ein USER-Port-Modul lieferbar.

Die Steuerung des Datenverkehrs und der Datenaustausch selber können nun aus einer Assemblerroutine oder aus einem vom Anwender geschriebenen BASIC-Programm heraus vorgenommen werden.

Für das Einlesen von Daten stellt der BASIC-Interpreter die Funktion INP (X) zur Verfügung. Die Variable X, auch Konstante, muß eine Integerzahl von 0 bis 255 sein. Mit ihr wird die Adresse definiert, unter der Daten von einem der Peripheriegeräte eingelesen werden sollen (9.3).

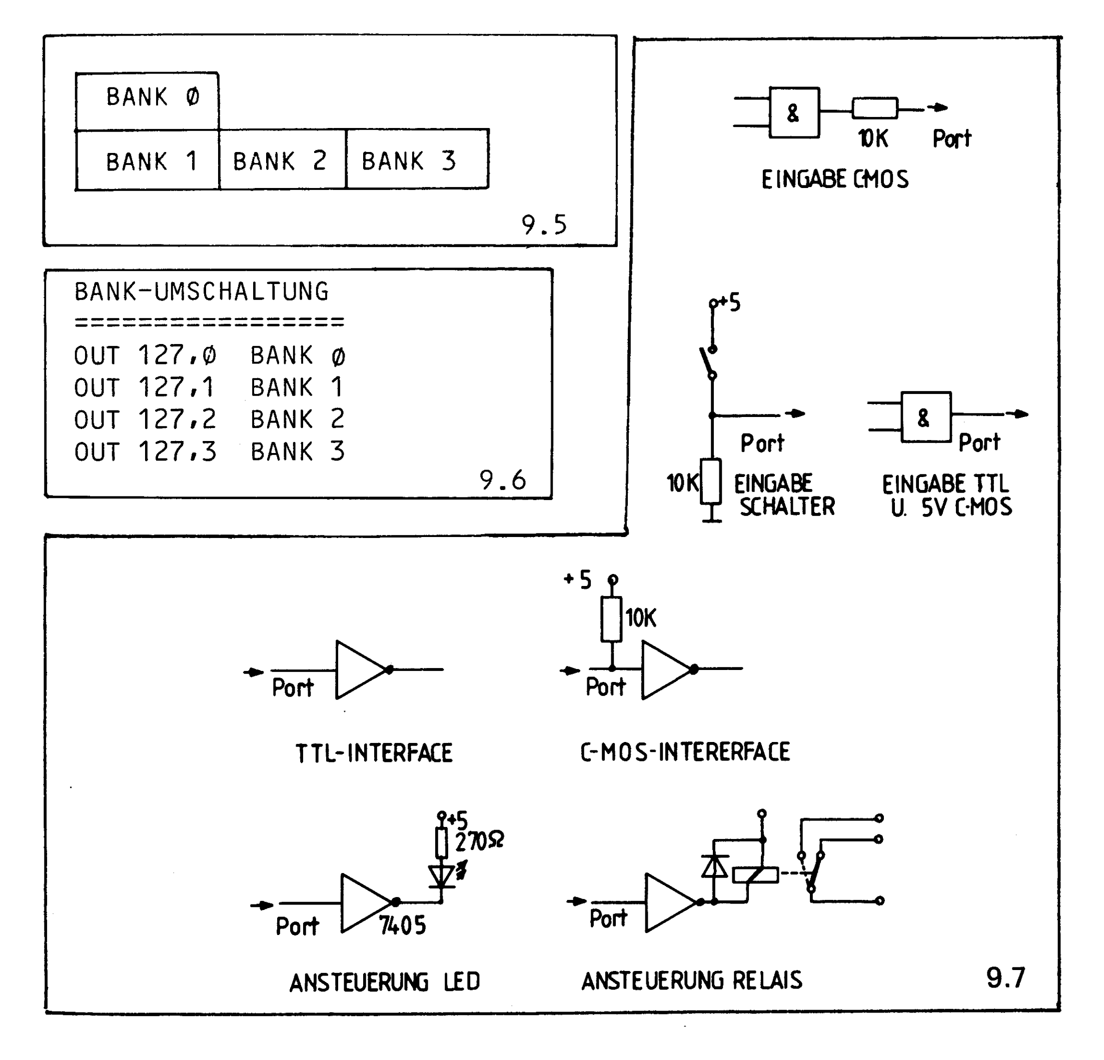
Mit der Anweisung OUT X,Y werden aus dem BASIC-Programm Daten- und Steueranweisungen an das durch die Integerzahl X angesprochene Gerät übergeben. Die übermittelten Daten im Bereich von 0 bis 255 werden hier durch die Variable Y dargestellt. Für X und Y können auch Konstanten oder Funk-

tionen zur Beschreibung der Werte genannt werden (9.4).

Ein Beispiel für das Einlesen von Daten aus einem Peripherie-Gerät zeigt schon das Kapitel zur Programmierung der Joysticks.

Die 64 KB RAM-Erweiterung

Die 64 KB RAM-Erweiterung belegt einen Speicherbereich von 32 KB. Sie ist in vier Speicherbänke von je 16 KB aufgeteilt (9.5). Nach dem Einschalten des Gerätes wird der zur Verfügung stehende RAM-Bereich von den Speicherbänken '0' und '1' belegt. Dadurch stehen dem BASIC-Programmierer insgesamt 32 KB RAM-Bereich zur Verfügung. Ein größerer Speicherbereich ist physikalisch nicht möglich, außerdem kann auch der BASIC-Interpreter nicht mehr als 32 KB RAM verwalten.



Die Benutzung der Speicherbänke '2' und '3' in einem BASIC-Programm ist problematisch, zeigt doch Anhang 11, daß dann im Bereich der Bank '1' der Zeichenkettenbereich und der BASIC-Stack liegen würden. Wird nun die Bank '1' durch die Bank '2' ersetzt, so verliert das BASIC-Programm alle als Zeichenketten abgelegten Daten, und die im Stackbereich verlorenen Informationen über GOSUB-Rückkehradressen und aktive FOR-NEXT-Schleifen werden zu einem Zusammenbruch des BASIC-Interpreters führen.

Trotzdem kann ein BASIC-Programmierer die vorhandenen Bänke nutzen. Denkbar wären in den Bänken '2' und '3' Assemblerroutinen und Datenspeicher, die aus dem BASIC-Programm mit POKE beschrieben und mit PEEK zurückgelesen werden. Durch Versetzen des Zeigers 'Top Of Memory' (s. Kapitel 12) kann das BASIC-Programm in der Bank '0' laufen.

Unter Kontrolle einer ML-Routine (Machine Language) lassen sich alle Bänke problemlos nutzen. Sinnvoll ist es, in einer Bank das Hauptprogramm abzulegen und die anderen Bänke für die Ablage von Unterprogrammen und Daten zu nutzen. Es muß nur jeweils der Umschaltmechanismus kontrolliert eingesetzt werden.

Die Bankumschaltung ist als 'Write Only Latch' an den Peripheriebus angeschlossen. Dementsprechend erfolgt die Umschaltung in BASIC-Programmen mit der OUT-Anweisung (9.6).

Der USER-Port

Der User-Port ist Gerät 16 am Peripheriebus. Er ist eine universelle Schaltung zur Steuerung von Elektroniken aller Art und besteht in der Hauptsache aus einem Multifunktions-Chip vom Typ 8255. Vierundzwanzig Leitungen lassen sich unter Programmkontrolle als Eingänge oder als Ausgänge benutzen. An sie lassen sich TTL- oder CMOS-Schaltungen anschließen und Transistoren, LED's und Relais ansteuern. Verbunden mit der Elektronik einer Modelleisenbahn oder angeschlossen an Alarmanlage, Heizungssteuerung oder andere Elektroniken, kann das BASIC-Programm Steuerung und Auswertung übernehmen. (9.7) zeigt Interface-Möglichkeiten. Die folgenden Seiten zeigen mit Schaltungsbeispielen die Programmgestaltung und Anwendung.

Der USER-Port wird unter der Adresse 255 für die vorgesehene Anwendung eingestellt. Die 24 Ein-/Ausgabeleitungen sind in drei Gruppen zu je 8 Leitungen organisiert: Port A, Port B und Port C. Durch Eintrag in das Kontrollregister 255 wird festgelegt, welcher Port als Eingang und welcher als Ausgang arbeiten soll. Unter drei weiteren Adressen können diese Ports nun gesetzt oder gelesen werden (9.8).

```
OUT 255,X   KONTROLLREGISTER SETZEN

OUT 254,X   PORTC SCHREIBEN
INP (254)         LESEN

OUT 253,X   PORTB SCHREIBEN
INP (253)         LESEN

OUT 252,X   PORTA SCHREIBEN
INP (252)         LESEN
```

9.8

9.1 Digital/Analog-Wandler

Es wird eine sägezahnförmige Ausgangsspannung erzeugt.

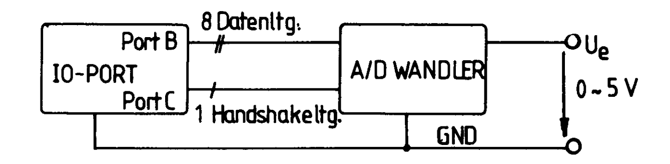
```
10 OUT 255,139                  :'KONTROLLREGISTER PORT A AUSGANG
20 FOR I=Ø TO 255               :'RAMPE IN 256 SCHRITTEN
30 OUT 252,I                    :'AN D/A-WANDLER VON PORT A
40 NEXT I
50 GOTO 20                      :'NAECHSTER IMPULS
```

IO - PORT A
8 Datenltg.
D/A WANDLER
Ua
GND
0

9.2 8-Bit Analog/Digtial-Wandler

Es werden kontinuierlich analoge Werte zu digitalen Werten gewandelt und angezeigt.

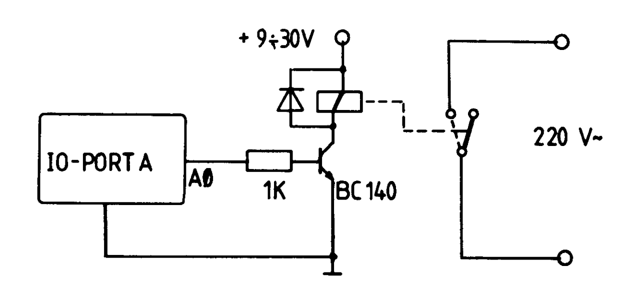
```
10 'PORT B INPUT - PORT C0-C3 OUTPUT - PORT C4-C7 INPUT
20 'PORT A OUTPUT
30 OUT 255,138                  :'PORTS KONFIGURIEREN
40 OUT 254,1: OUT 254,Ø         :'START OF CONVERSION
50 PRINT INP (253)              :'WERT ANZEIGEN
60 GOTO 40                      :'NAECHSTEN WERT LESEN
```



9.3 Ein-/Ausschalten von Relais zur Steuerung beliebiger Verbraucher

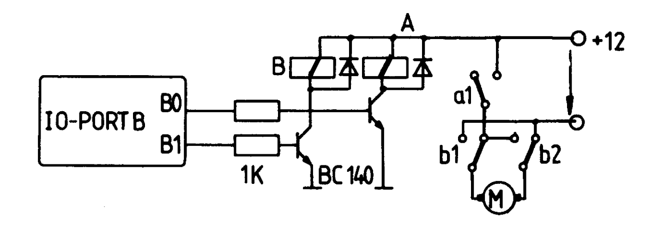
Eine Lampe wird ein- oder ausgeschaltet.

```
10 OUT 255,139                      :'PORT A IST AUSGANG
20 'EINSCHALTEN MIT OUT 252,1
30 'AUSSCHALTEN MIT OUT 252,Ø
```



9.4. Ein 12V-Motor wird in den Funktionen EIN, AUS, Rechtslauf und Linkslauf gesteuert.

```
10 OUT 255,153                     :'PORT B AUSGANG
20 'MOTOR RECHTSLAUF EIN   OUT 253,1
30 'MOTOR LINKSLAUF  EIN   OUT 253,3
40 'AUSSCHALTEN            OUT 253,Ø
```



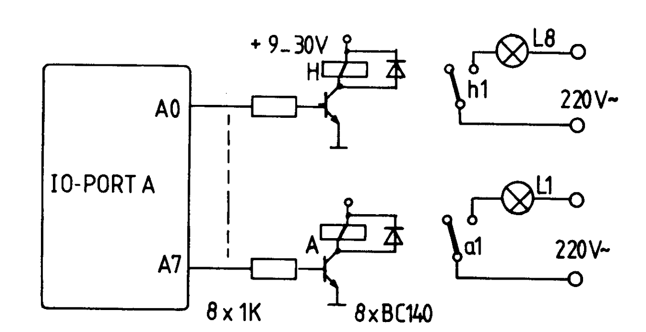
9.5 Lauflicht mit 8 Lampen

```
10 OUT 255,139                      :'PORT A AUSGANG
20 L(1)=1: L(2)=2: L(3)=4           :'BITMUSTER FUER AUSGABE
22 L(4)=8: L(5)=16: L(6)=32
24 L(7)=64: L(8)=128
30 FOR I=1 TO 8                     :'SCHLEIFE ZUM SCHALTEN
40 OUT 252,L(I)
```

```
50 FOR V=1 TO 1000: NEXT V      :'EINSCHALTDAUER
60 NEXT I                       :'NAECHSTE LAMPE
70 GOTO 30
```

ANMERKUNG:
FUER LINKSLAUF IST ZEILE 30 WIE FOLGT ZU ÄNDERN:

```
30 FOR I=8 TO 1 STEP -1
```



9.6 Ansteuerung eines Vier-Phasen-Schrittmotors

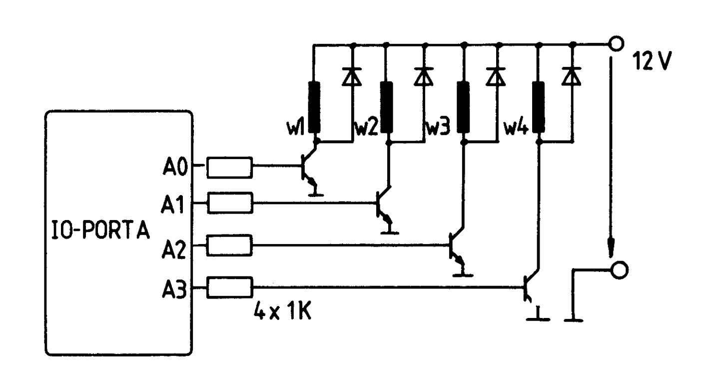
```
10 OUT 255,139                  :'PORT A IST AUSGANG
20 W(1)=1: W(2)=2               :'BITMUSTER DEFINIEREN
22 W(3)=4: W(4)=8
30 FOR I=1 TO 4
40 OUT 252,W(I)                 :'MOTORWICKLUNGEN SCHALTEN
50 FOR V=1 TO 250: NEXT V       :'GESCHWINDIGKEIT
60 NEXT I: GOTO 30
```

FUER LINKSLAUF IST ZEILE 30 WIE FOLGT ZU AENDERN:

```
30 FOR I=4 TO 1 STEP -1
```

FUER HALBSCHRITTBETRIEB WIRD WIE FOLGT GEAENDERT:

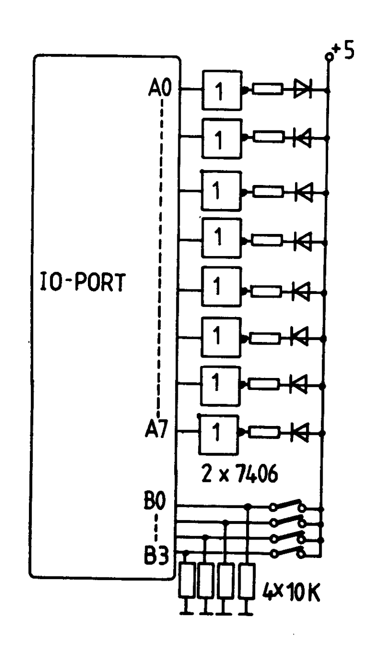
```
20 W(1)=1:W(2)=3:W(3)=2:W(4)=6:W(5)=4:W(6)=12:
   W(7)=8:W(8)=9
30 FOR I=1 TO 8
```



9.7 Rechts/Links-Lauflicht mit Schaltersteuerung

KEIN SCHALTER 'EIN': RECHTSLAUF, EIN BELIEBIGER SCHALTER 'EIN': LINKSLAUF.

```
10 OUT 255,130              :'PORT A=AUSGANG, B=EINGANG
20 A=INP(253)               :'SCHALTER ABFRAGEN
30 IF A=0 THEN 200          :'KEIN SCHALTER: RECHTSLAUF
100 FOR I=128 TO 1 STEP-1   :'SONST LINKSLAUF
110 OUT 252,I               :'I IST BITMUSTER 1000 0000
120 I=I/2+1                 :'BIT SCHIEBEN, LOOP VAR KORR.
130 NEXT I:GOTO20           :'NAECHSTE LED AN
200 FOR I=1 TO 128          :'RECHTSLAUF
210 OUT 252,I               :'BITMUSTER IST ERST 0000 0001
220 I=I*2-1                 :'BIT RECHTS SCHIEBEN
230 NEXTI: GOTO 20
```



10 Programmieren der Joysticks

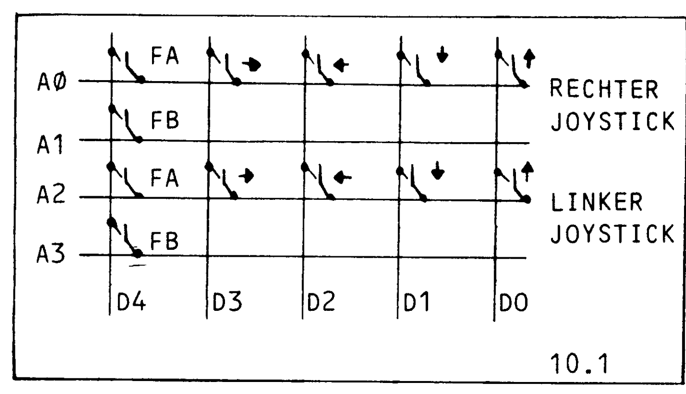
mit INP (X), ON...GOTO..., ON...GOSUB...

Die Organisation der Joysticks

Die Schalter der Joysticks sind in einer Matrix angeordnet (10.1). Über die Adressenleitungen A0 bis A2 werden die Zeilen der Matrix je nach ausgegebener Adresse auf '0'-Potential gelegt. Tabelle (10.2) gibt Aufschluß über die Adressen.

Mit der Anweisung 'INP (adresse)' kann der Zustand der Datenleitungen gelesen werden. Je nach angegebener Adresse werden die Datenleitungen D0 bis D4 Aufschluß darüber geben, welcher Schalter im Moment des Einlesens betätigt war.

Diese Information, mit INP (adresse) ermittelt, wird in codierter



10.1

ADRESSE		A3	A2	A1	A∅
27H	39	0	1	1	1
2BH	43	1	0	1	1
2DH	45	1	1	0	1
2EH	46	1	1	1	0

10.2

27H	39	TASTER FB LINKER JOYSTICK
2BH	43	TASTER FA, RICHTUNG LINKER JOYSTICK
2DH	45	TASTER FB RECHTER JOYSTICK
2EH	46	TASTER FA, RICHTUNG RECHTER JOYSTICK

10.3

Form als dezimale Zahl vom BASIC-Interpreter bearbeitet. In der Form A=INP (adresse) wird sie in einer Variablen abgelegt.

Tabelle (10.3) stellt den Zusammenhang zwischen Adresse und Funktion her. Sollen die Richtungsinformationen des rechten Joysticks ausgewertet werden, so ist die Anweisung 'A= INP (46)' in das Programm aufzunehmen. Entsprechend ist für die anderen Funktionen zu verfahren.

Die mit der INP-Funktion ermittelte Zahl liegt im Bereich von 0 bis 255, entsprechend den 8 Datenleitungen D0 - D7. Da nur die Datenleitungen D0 bis D4 benutzt werden, wird der mit INP (adresse) gelesene Wert mit einer MASKE in einer AND-Funktion verknüpft. Die Datenbits D5 bis D7 des Wertes der Variablen werden definiert auf '0' gesetzt, die informationstragenden Bits D0 bis D4 werden in ihrem Wert nicht beeinflußt (10.4).

Die vier Anweisungen zum Lesen der Joysticks sind also:

```
A = INP (39) AND 31  JOY LINKS,  TASTER FB
A = INP (43) AND 31  JOY LINKS,  TASTER FA, RICHTUNG
A = INP (45) AND 31  JOY RECHTS, TASTER FB
A = INP (46) AND 31  JOY RECHTS, TASTER FA, RICHTUNG
```

```
D7 D6 D5 D4 D3 D2 D1 DØ
-  -  -  X  X  X  X  X   INP(adr)
0  0  0  1  1  1  1  1   AND 31
--------------------------------
0  0  0  X  X  X  X  X   Ergebnis
                         in A
                          10.4
```

Tabelle 10.5. stellt den Zusammenhang der codierten Informationen, die in der Variablen 'A' hier abgelegt sind, und den Joystick-Funktionen her.

Mit Hilfe von AND-Masken lassen sich Einzelinformationen unter den beiden Adressen 43 und 46 (Richtung und Taster A) lesen. Ein Beispiel: Eine Variable soll nur die Informationen Joystick rechts, Taster FA gedrückt oder nicht gedrückt, enthalten. Zeichnung (10.1) zeigt Taster FA an der Datenleitung D4. Tabelle (10.3) ergibt die Adresse für Joystick rechts, Taster FA. Die Leseanweisung wäre also: A = INP(46). Die Maske würde entsprechend (10.4) binär 0001 0000 sein, dezimal wird

sich der Wert 16 ergeben. Programm (10.6) zeigt nach RUN, daß der Taster FA den Wert 0 ergibt, sonst wird das Ergebnis stets 16 sein. Es sind also alle Funktionen außer Betrieb (=16), nur Taster FA zeigt bei Betätigung den Wert 0.

Die Umrechnung von binären Zahlen zu dezimalen Zahlen kann mit der Tabelle (10.7) durchgeführt werden. Für alle auf '1' gesetzten Bits einer achtstelligen Binärzahl ist der Wert entsprechend der Tabelle zu addieren. Beispiel: 1010 0011 binär wird dezimal 1+2+32+128 = 163 ergeben.

ADRESSE JOY LI	RE	FUNKTION	RESULTAT a	b
39	45	TAST. FB	15	-5
43	46	LI,AUFW	26	6
		LI,ABW	25	5
		RE,AUFW	22	2
		RE,ABW	21	1
		AUFW.	30	10
		ABW.	29	9
		LINKS	27	7
		RECHTS	23	3
		TAŚT.FA	15	-5
		AUS	31	11

a) A=INP(X) AND 31
b) A=(INP(X) AND 31)-20

10.5

```
10 A = INP(46) AND 16
20 PRINT A: GOTO 10
```

10.6

BINÄRE WERTIGKEIT

BIT	WERT
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

10.7

ON X GOTO ..., ON X GOSUB ...

Für die Auswertung der Richtungsadressen bietet sich die BASIC-Funktion 'ON variable GOTO liste,liste' an. Das LASER-BASIC aber kennt das Schlüsselwort ON nicht. Da die ausführenden Routinen aber im Interpreter vorhanden sind, kann, wie in Kapitel 3, als erste Zeile des Programmes, die ON-Anweisung in der richtigen Schreibweise eingegeben werden. Anstelle des 'ON'-Wortes ist ein anderes Schlüsselwort zu schreiben, in (10.8) ist es PRINT, dessen Code mit einer POKE-Anweisung von der Tastatur zum Code für ON verändert wird. Wird vom Wert der INP-Funktion für die Richtungsadressen 20 abgezogen, so wird das Ergebnis im Bereich von -5 bis 11 liegen. -5 für Taster FA kann mit IF ausgeschlossen werden. Der Code für alle 8 Richtungen reicht dann von 1 bis 10. 11 ergibt sich für 'keine Taste' gedrückt. Damit liegt er im Bereich der ON-Funktion, die als Sprungverteiler abhängig vom Inhalt der nach ON genannten Variablen arbeitet. Die Variable muß Integer-Zahlen von 1 aufwärts enthalten. Nach GOTO oder GOSUB folgt dann eine Liste von Zeilennummern. Mit der Variablen =1 wird zur ersten Zeile der Liste verzweigt, mit =2 zur zweiten usw. (10.8).

Programm (10.9) wendet die ON-Funktion in der geschilderten Form an. Es ist als Demonstrationsprogramm für die Joysticks gedacht. Je ein Punkt wird im Grafik-MODE(1) über den Bildschirm bewegt. Es lassen sich Zeichnungen auf dem Schirm anfertigen. Taster FB bewegt zusammen mit den Steuerknüppeln den Punkt, ohne zu zeichnen.

```
2 PRINT A GOTO 100,200,300
(1. ZEILE IM PROGRAMM)
POKE 31469,161 <RETURN>
(OHNE ZEILENNUMMER!)
                              10.8
```

```
1 GOTO3: '2 ON A GOTO 110,120,130,,150,160,170,,190,200,210
2
3 CLS: COLOR2,1 :DIM RI(3),AD(3),W(3): RESTORE
4 FOR I=0 TO 3: READ AD(I),W(I): NEXT: MODE (1)
10 SET (W(0),W(1)): SET (W(2),W(3))
20 FOR I=0 TO 3: RI(I)=(INP(AD(I)AND31)-20: NEXT
30 IF RI(0)=-5 RESET(W(0),W(1))
31 IF RI(2)=-5 RESET (W(2),W(3))
40 C=0: A=RI(1): IF A>0 GOSUB 2
```

```
41 C=2: A=RI(3): IF A>0 GOSUB 2
50 GOTO 10
110 W(0+C)=W(0+C)+1: W(1+C)=W(1+C)+1: GOTO 202
120 W(0+C)=W(0+C)+1: W(1+C)=W(1+C)-1: GOTO 202
130 W(0+C)=W(0+C)+1: GOTO 202
150 W(0+C)=W(0+C)-1: W(1+C)=W(1+C)+1: GOTO 202
160 W(0+C)=W(0+C)-1: W(1+C)=W(1+C)-1: GOTO 202
170 W(0+C)=W(0+C)-1: GOTO 202
190 W(1+C)=W(1+C)+1: GOTO 202
200 W(1+C)=W(1+C)-1
202 IFW(0+C)>127THENW(0+C)=127
204 IF W(0+C)<0 THENW(0+C)=0
206 IFW(1+C)>63 THEN W(1+C)=63
208 IFW(1+C)<0  THENW(1+C)=0
210 RETURN
1000 DATA39,0,43,0,45,127,46,63
                                                 10.9
```

BASIC-UP

(10.10) zeigt die Anwendung der ON-Funktion, wenn vorher das BASIC-Erweiterungsprogramm 'BASIC-UP' geladen wurde.

```
■ON A GOTO 100,200,300
■ON A GOSUB 100,200,300
                          10.10
```

11 Maschinennahe Programmierung

Erstellen von Programmen - Speicherbereitstellung - Schutz vor BASIC - CSAVE, CLOAD und CRUN für ML-Programme

11.1 Einleitung

Es wurde nun schon vieles über den Gebrauch des LASER-Computers als BASIC-Computer geschrieben, und es wird fast selbstverständlich erscheinen, daß er BASIC versteht. Dies ist aber nur durch ein Zusammenwirken der Hardware (d.h. der elektronischen Bauteile) und der Software (den Programmen BASIC-Interpreter und Betriebssystem) zu erreichen. Alle Befehle, die in BASIC eingegeben werden, sei es direkt oder als Programm, und jede Tasteneingabe werden von dieser Kombination von Software und Hardware bearbeitet. Daher scheint es, als verstünde das Gerät BASIC wie eine Sprache.

Als Vergleich: Ein eigenes BASIC-Programm ist mit RUN gestartet, und der

Rechner versteht keinen BASIC-Befehl mehr, nur noch die im Programm vorgesehenen Kommandos. Das Programm:

```
10 PRINT "SYNTAX ERROR": PRINT "READY": INPUT BEFEHL$
20 IF BEFEHL$ <> "BASIC" THEN GOTO 10
```

demonstriert dies in einer krassen Weise: Nachdem es gestartet wurde, nimmt der Rechner zwar Eingaben an, reagiert aber nur auf den 'Befehl' BASIC - er schaltet BASIC ein.

Ein BASIC-Programm kann jederzeit mit BREAK unterbrochen werden. In diesem Kapitel wird erläutert, welche Möglichkeiten bestehen, den BASIC-Interpreter zu unterbrechen und dann die Maschine direkt zu beeinflussen.

11.2 Aufbau des Rechners

Der Aufbau des LASER-Computers entspricht prinzipiell dem aller heutigen Mikrocomputer. Der Zentralprozessor vom Typ Z 80 übernimmt die Befehlsausführung. Die Kommandos werden aus dem Speicher entnommen, aus dem teils nur gelesen werden kann (die ROM-Bausteine) und in die zum Teil auch geschrieben werden kann (die RAM-Speicher). Die ROM's sind Speicherbausteine, in denen die gespeicherten Informationen auch nach dem Abschalten erhalten bleiben. In ihnen befinden sich der BASIC-Interpreter und das Betriebssystem. In den RAM-Speichern können Daten und Programme untergebracht werden. Im Gegensatz zu den ROM-Speichern sind diese Informationen nach dem Abschalten der Betriebsspannung aber verloren.

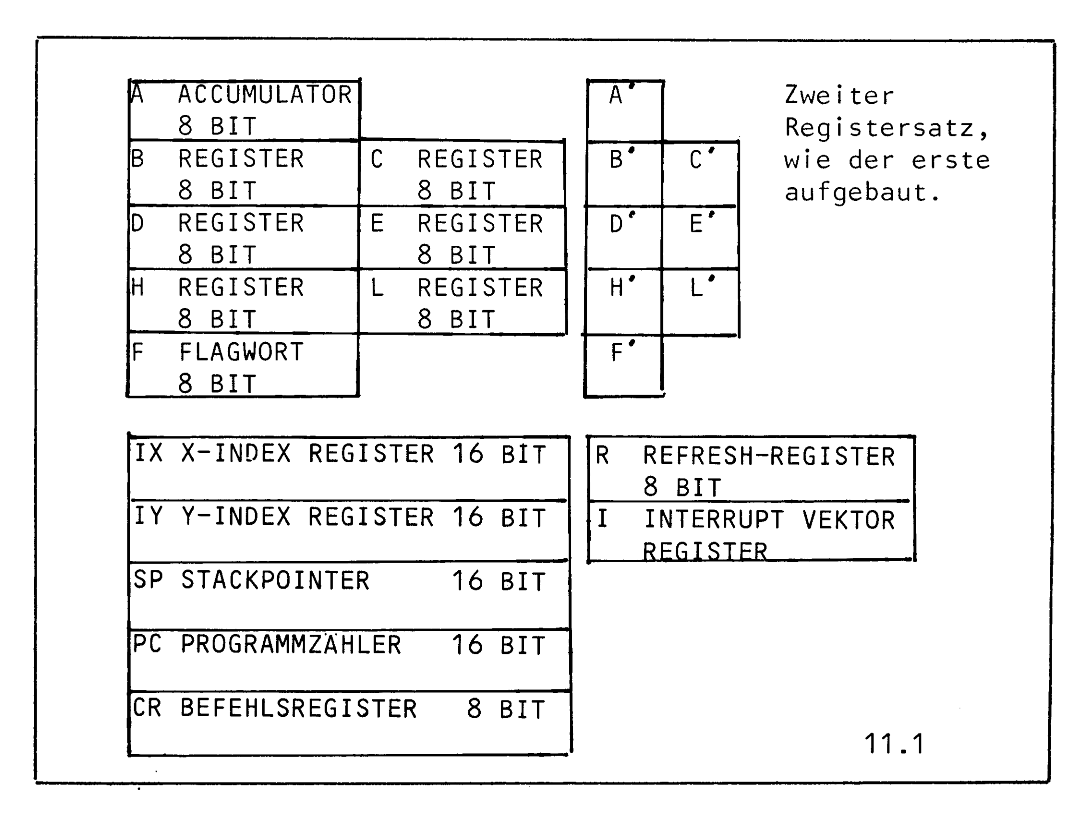
Der Z 80 Mikroprozessor kann nun eine Reihe von Anweisungen ausführen, die insgesamt die sogenannte Maschinensprache darstellen. Zur Einarbeitung in die maschinennahe Programmierung sollte die in der einschlägigen Literatur zu findende Beschreibung von Funktion, Anwendung und Programmierung durchgearbeitet werden. Hier kann nur ein kurzer Überblick gegeben werden.

11.3 Aufbau und Funktion

Für die weiteren Darstellungen in diesem Kapitel sollten die Ausdrücke Bit, Byte und hexadezimal bekannt sein. Um Programme in Maschinensprache verarbeiten zu können, verfügt die Z 80 CPU (Central Processing Unit - Zentral-Prozessor, ZP) über eine Anzahl von internen Speichern, sogenannte Register. Die übliche Darstellung zeigt (12.1).

Der Ablauf einer Befehlsausführung läßt sich nun so beschreiben: Der Inhalt des Programmzählers wird als Adresse an die Speicherbausteine über den Adreßbus (16 Leitungen) ausgegeben. Der an diesem so bezeichneten Speicherplatz befindliche Code wird in das Befehlsregister transportiert und der Programmzähler um 1 erhöht. Je nach Befehlscode werden nun unter Kontrolle der CPU Werte aus den folgenden Speicherstellen in die Register geladen, bearbeitet, verknüpft und

berechnet und neue Operationen vorbereitet. Entsprechend wird der Programmzähler erhöht oder verändert. Ein Maschinenbefehl besteht deshalb aus 1 bis 4 Speicherworten, die aufeinander folgen. (Ein Speicherwort = ein Byte = 8 Bit. Es kann 256 verschiedene Informationen darstellen, die sich durch die Zahlen von 0 bis 255 ausdrücken lassen.) Solange keine Sprünge erfolgen, werden die Befehle und Daten aus aufeinander folgenden Speicherstellen in die CPU transportiert.



11.1

11.4 Übergang von BASIC auf Maschinensprache

Um ein Maschinenprogramm in die Speicher zu laden, benutzt man unter Kontrolle des BASIC-Interpreters den POKE-Befehl. Die den Maschinenbefehlen und ihren Operanden entsprechenden dezimalen Zahlen werden mit POKE 'adresse, wert' fortlaufend in die Speicher geschrieben.

Bei der Festlegung der Startadresse ist dafür zu sorgen, daß der vorgesehene Speicherbereich vor Zugriffen des BASIC-Interpreters geschützt wird (siehe hierzu Abschnitt 6). Anschließend wird die Startadresse des Programmes in den USR-Vektor eingetragen. Das kann durch das Programm (11.2) geschehen. Es ist dazu durch die Anweisungszeilen in (11.3) zu ergänzen.

Das ML-Programm kann dann mit X = USR(X) gestartet werden. Nach Beendigung des ML-Programmes mit 'RET' erfolgt die Rückkehr in das BASIC-Programm, das mit der nächsten folgenden Anweisung fortgesetzt wird.

```
Ein Beispiel: (11.2)

 5 ADRESSE =  29184          :'Startadresse des ML-Prgr.
20 READ WERT%                :'Ersten Wert aus ML-Prgr.-Liste
30 IF WERT%=-1 THEN END      :'Ende der Liste
40 POKE ADRESSE,WERT%        :'In den Speicher eintragen
50 ADRESSE = ADRESSE +1      :'Adressiere nächste Speicherst.
60 GOTO 20                   :'In Schleife weiterarbeiten
70 '---------------------------------------------------------
100 DATA ...,...,...,...,    :'ML-Progr. in dezimalen Werten
120 DATA ...,...,...,...,    :'Ende der Liste mit ...,-1
```

```
10 HB% = AD/256          :'MSB DEZIMAL ERRECHNEN
12 LB%= AD-(HB%*256)     :'LSB ERRECHNEN
14 POKE 30863,HB%        :'VEKTOR MSB SETZEN
16 POKE 30862,LB%        :'VEKTOR LSB SETZEN

                                            11.3
```

11.5 Der USR-Vektor

Im vorstehenden Programm wurde für den Aufruf eines Maschinenprogrammes aus einem BASIC-Programm heraus dieser Vektor gesetzt. Hier nun einige weitere Informationen zum USR-Befehl: Nach dem Einschalten und Initialisieren des BASIC-Interpreters ist dieser Vektor auf die Fehler-Ausgaberoutine gerichtet. Ein Aufruf des ML-Programmes führt so zur

Fehlermeldung ?FUNCTION CODE ERROR IN XX.

Ist dieser Vektor nun ordnungsgemäß auf die Adresse der Anwender-Routine gerichtet, so kann die Kontrolle mit X = USR (X) an dieses Programm übergeben werden. Im einfachsten Falle ist in dieser Funktion X eine Variable, deren Inhalt im weiteren Programmlauf keine Rolle spielt, also eine DUMMY-Variable. In anderen Anwendungsfällen kann mit dieser Funktion dem Maschinenprogramm ein Parameter übergeben werden, der innerhalb der Klammern als Konstante oder Variable jeder Art, auch Zeichenkette, definiert werden kann. Die Variable wird mit dem Aufruf der USR-Funktion an das BASIC-Register WRA1 übergeben. Wird eine String-Variable oder Konstante übergeben, so ist in WRA1 ein Zeiger auf den String-Deskriptorblock zu finden (Länge und Adresse). Wird die Kontrolle des Programmlaufes mit RET als letzte Anweisung des ML-Programmes an BASIC zurückgegeben, so wird der Inhalt von WRA1 der Variablen im USR-Befehl zugewiesen und kann im BASIC-Programm weiter verarbeitet werden.

Das folgende BASIC-Programm richtet den USR-Vektor auf die 'BEEP'-Routine und legt eine Zeichenkette über WRA1 in A$ ab (11.4).

Die Adressen für WRA1 und dem zugehörigen Variablen-Typflag sind dem Anhang 12 zu entnehmen.

```
10 POKE 30862,80 : POKE 30863,52 : 'USR-VEKTOR AUF BEEP
20 A$ = USR ("TEST")
30 PRINT A$
RUN
TEST
                                                    11.4
```

11.6 Warum Maschinensprache?

Prinzipiell bietet BASIC die selben Möglichkeiten, Programme zu schreiben wie mit einem Z 80-Assembler. Der Verzicht auf die Funktionen des BASIC-Interpreters hat Vor- und Nachteile.

Die schnelle Ausführung von Maschinenprogrammen ist wohl der Hauptvorteil. Das folgende BASIC-Programm invertiert den Bildschirm (Zeilen 20 bis 40). Die Zeile 60 legt ein Maschinenprogramm in den von Zeile 10 reserviertem Bereich ab und startet es in Zeile 70 - 80. Der vorher von BASIC invers geschaltete Bildschirm wird jetzt mit der Geschwindigkeit eines ML-Programmes wieder auf normale Darstellung geschaltet. Der Unterschied in der Ausführungs-Geschwindigkeit wird so mehr als deutlich.

```
5 PRINT"BASIC-START"
10 REM12345678901234567890 :'RESERVIERT FÜR ML-PROGRAMM
20 FOR I%=28672 TO 29183   :'BILDSCHIRM INVERS SCHALTEN
30  A%=PEEKK(I%)           :'MIT BASIC-GESCHWINDIGKEIT
40  B%=A% AND 64
50  IF B% THEN A%=A% AND 191 ELSE A%=A% OR 64
60  POKE I%,A%
70 NEXT I%
80 FOR I%=31493 TO 31505   :'ML-PROGRAMM AUS DATA-ZEILEN
90 READ A%                 :'LESEN UND ...
100 POKE I%,A%             :'IN REM-ZEILE EINTRAGEN
110 NEXT I%
120 POKE 30862,5           :'USR-VEKTOR LOW SETZEN
130 POKE 30863,123         :'UND HIGH BYTE SETZEN
140 PRINT "ML-START"       :'ML-PROGRAMM...
150 A%= USR (0)            :'STARTEN
160 END
```

```
200 DATA 33,0,112,1,0,2,126,238,64,119
210 DATA 35,11,120,177,32,246,201
                                    11.5
```

Dieser Unterschied resultiert aus der 'Pingeligkeit' des BASIC-Interpreters, der die meiste Zeit damit verbringt, Fehler zu suchen und dem Anwender die Programmierung zu erleichtern; indem er vorbereitete Funktionen zur Verfügung stellt.

Der Zeitvorteil der ML-Programmierung wird damit erkauft, daß der Anwender ohne Schutz vor möglichen Programmfehlern ist. Wenn das Maschinenprogramm in (11.5) mit POKE 31509,0 fehlerhaft gemacht und mit A=USR(0) gestartet wird, kann beobachtet werden, was ein fehlerhaftes Programm anrichtet.

11.7 Speicherplatz für Maschinenprogramme

a) Reservierung im BASIC-Text

Wie das Beispiel (11.5) zeigt, können kurze ML-Routinen innerhalb des BASIC-Programmes in durch REM-Befehl reservierten Speicherplatz geladen werden. Das ML-Programm darf dann aber nicht länger als eine BASIC-Zeile werden. Ist das ML-Programm geladen, wird es Schwierigkeiten beim Auflisten des Programmes geben. Es sollte in der Phase der Editierung also besonders sorgfältig gearbeitet werden und vor dem ersten Test der ML-Routine das BASIC-Programm auf Kassette gesichert sein.

b) Reservierung im Video-RAM

Für den Bildwiederholspeicher stehen 2 KB RAM zur Verfügung. Sie werden im Adressenbereich 7000H bis 7FFFH decodiert. Im Textbetrieb ('MODE (0)') des Video-Kontrollers werden nur 512 Bytes für die Abbildung auf dem Bildschirm benutzt, der restliche Bereich steht für Anwender-Routinen zur Verfügung. Da es bei unkontrollierten Zugriffen des Mikroprozessors auf das Bildschirm-RAM zu Störungen im Bild kommen kann, sollten hier hauptsächlich Interrupt-Routinen und Bildschirm-Informationen abgelegt werden.

c) Ablage im 'Free Space'-Bereich

Anhang 11 zeigt, daß es bei laufendem BASIC-Programm zwischen dem Ende der Variablentabellen und dem Ende des String-Bereiches einen freien Speicherbereich gibt. Auch hier können ML-Routinen abgelegt werden. Allerding ist sicherzustellen, daß das hier abgelegte Programm nicht von dem sich dynamisch ändernden BASIC-Stack überschrieben werden kann.

d) Ablage vor dem BASIC-Text

Wird der Beginn des BASIC-Textes zu höheren Bereichen verschoben, so kann zwischen Ende der System-Variablen und dem Anfang des BASIC-Textes Platz für Maschinenprogramme geschaffen werden, die vor allen Aktivitäten des BASIC-

```
POKE 32768,0 <CR>      :'ANHANG 16, DREIMAL '0'
POKE -32767,0 <CR>
POKE -32766,0 <CR>
POKE 30884,1 <CR>      :'BASIC-START-ZEIGER SETZEN
POKE 30885,128 <CR>    :'AUF 8001H, 32769
POKE 30969,3 <CR>      :'VARIABLEN-START-ZEIGER SETZEN
POKE 30970,128 <CR>    :'AUF 8003H, 32771
                                                    11.6
```

Interpreters geschützt sind. Ein BASIC-Programm, das dann geschrieben und auf Band gesichert wird, wird beim Laden von der Kassette automatisch wieder an den selben Platz abgelegt. Das notwendige Umsetzen der Zeiger geschieht durch POKE-Anweisungen direkt von der Tastatur. Im folgenden Beispiel soll der Start des BASIC-Textes von üblicherweise 7AE8H nach 8000H verschoben werden (siehe auch Anhang 11 und 16). Es sind der Reihe nach die folgenden Anweisungen zu geben (11.6):

e) Die sicherste Methode: Am Speicher-Ende!

Die am häufigsten angewandte Methode ist, den Zeiger 'Top Of Memory', der auf die letzte RAM-Zelle zeigt, durch zwei POKE-Kommandos umzusetzen. Dieses kann im BASIC-Programm auch als erste Anweisung geschehen. Wird danach CLEAR XX gegeben, um Platz für Zeichenketten zu reservieren, werden alle vom TOM abhängigen Zeiger des Betriebssystems automatisch neu gesetzt. Die Anweisung:

```
10 POKE 30898, PEEK(30898)-2: CLEAR 1000
```

reserviert 2*256 Bytes für ML-Programme, setzt die Zeiger des String-Bereiches und den BASIC-Stackpointer neu und reserviert 1000 Bytes für die Ablage von Zeichenketten. Es ist wichtig, daß CLEAR mit Parameter gegeben wird, da sonst die Zeigerkorrektur nicht durchgeführt wird. Notfalls ist CLEAR 50 oder auch CLEAR 1 zu geben.

11.8 Abspeichern der ML-Programme auf Kassette

Um eine vom Anwender entwickelte ML-Routine auf Kassette abzuspeichern, ist das Programm (11.8) einzugeben. Wie vorher diskutiert, ist für diese Routine (wie für das Anwenderprogramm) entsprechend RAM-Platz zu reservieren und das Ladeprogramm mit Setzen des USR-Vektors und Aufruf mit X=USR(X) zu starten.

11.9 Anwender Interrupt-Vektor

Ein Interrupt-Zeiger unter der Adresse 787DH im RAM ermöglicht es, vom Anwender geschriebene Routinen in den System-Interrupt einzubinden. Die einzige Interrupt-Quelle ist der Video-Display-Generator, der alle 20 msek einen Inter-

```
DI                          ;DISABLE INTERRUPT
LD C,F1H                    ;KENNZAHL FÜR BINÄR-PROGRAMM
LD HL,(78A4H)               ;BASIC-START-POINTER RETTEN
PUSH HL
LD HL,(78F9H)               ;BASIC-END-POINTER RETTEN
PUSH HL
LD HL,XXXXH                 ;HIER STARTADR. EINSETZEN
LD (78A4H),HL               ;ZEIGER AUF START BIN-DATEI
LD HL,YYYYH                 ;HIER ENDADR. EINSETZEN
LD (789FH),HL               ;ZEIGER AUF ENDE BIN-DATEI
LD HL,ZZZZ                  ;HIER ADR. DES PROGR.-NAMEN
CALL 34ACH                  ;UND ABSPEICHERN
POP HL                      ;BASIC-END-POINTER HERSTELLEN
LD (78F9H),HL
POP HL                      ;BASIC START-POINTER HERSTELLEN
LD (78A4H)
EI                          ;INTERRUPT ZULASSEN
RET                         ;RÜCKKEHR ZU BASIC
;
ZZZZ DEFM '"NAME"'          ;HIER PROGR.-NAME MAX. 15 ZEICHEN
     DEFB 00H               ;ABSCHLUSS NAME
                                                          11.8
```

```
LD HL,7AE9H                 ;STARTADR. BASIC
LD (78A4H),HL               ;WIEDER HERSTELLEN
JP 36CFH                    ;UND SPRUNG NACH BASIC-START
,                           ;HIER START DER ANWENDER ROUTINE
,
                                                          11.9
```

rupt für den Z 80 Prozessor am Anschluß INT auslöst. Es wird MODE1 des maskierten Interrupts benutzt. Da der den Interrupt auslösende Impuls der vertikale Synchron-Impuls des Video-Kontrollers ist, sollten Zugriffe auf das Video-RAM nur während der Sync-Austastlücke im Interrupt-Programm erfolgen. Störungen im Bild sind sonst nicht zu vermeiden. Wird von der CPU ein Interrupt-Impuls empfangen und ist die Ausführung zugelassen, so wird der Programmlauf bei 0038H fortgesetzt. (11.10) zeigt die Interrupt-Serviceroutine im ROM1 der LASER-Computer:

In 787DH ist nach dem Einschalten des Computers der Code des RET-Befehles eingetragen. Hier kann nun aus einer Anwender-Routine heraus ein JP-Befehl auf eine eigene Interrupt-Routine implementiert werden.

Die Anwender-Interrupt-Routine kann auch mit RET abgeschlossen werden, um eine Fortsetzung der ROM-Routine zu erreichen.

```
0038    JP 2EB8

2EB8    PUSH AF         ;INTERRUPT SERVICE ROUTINE
        PUSH BC         ;REGISTER RETTEN
        PUSH DE
        PUSH HL
        CALL 787DH      ;ANWENDER INTERRUPT
        CALL SCREEN     ;BILDSCHIRM AUFBAUEN
        CALL CURSOR     ;CURSOR BLINKEN LASSEN
        CALL KEYBRD     ;TASTATUR PRÜFEN
        CALL BUZZER     ;BEEP, WENN TASTE GEDR.
        POP HL          ;REGISTER WIEDER HERSTELLEN
        POP DE
        POP BC
        POP AF
        EI
        RETI            ;INTERRUPT BEENDEN
                                        11.10
```

```
        ORG 787DH               ;ANWENDER INT RAM-SPRUNG
INTRAM  DEFS 3                  ;DREI BYTES RESERVIEREN
START   EQU 7AE9                ;START ANWENDER-ROUTINE
        ;
        ;
        ORG START
        DI                      ;INTERRUPT ABSCHALTEN
        LD SP,7FFFH             ;STACK NEU INITIALISIEREN
        LD A,C3H                ;OBJ-CODE DER JP-ANWEISUNG
        LD (INTRAM),A           ;IN DEN RAM-SPRUNG-VEKTOR
        LD HL,USRINT            ;BEGINN ANWEND.INTERR.ROUTINE
        LD (INTRAM+1),HL        ;IN SPRUNG-ANWEISUNG
        EI                      ;INTERRUPT EINSCHALTEN
        ,                       ;WEITER MIT DEM
        ,                       ;...VORDERGRUND-PROGRAMM
USRINT  CALL KEYBRD             ;TASTATUR PRÜFEN
        ,                       ;ANWENDER INT-PROGRAMM
        ,
        POP HL                  ;RÜCKSPRUNGADR. VOM STACK
        POP HL                  ;REGISTER HERSTELLEN
        POP DE                  ;FÜR BEENDEN DER INT-ROUTINE
        POP BC
        POP AF
        EI                      ;INT EINSCHALTEN
        RETI
                                                11.11
```

12 Für den Assembler-Programmierer

Tastatur-Abfrage, CRUN/CLOAD, Character zum Bildschirm, String - Ausgabe, Compare-Symbol, Teste nächste Zeichen, Compare DE/HL, Test Variablen-Typ, Control-Codes, Joysticks, Buzzer-BEEP, Drucker-Steuerung

Dieses Kapitel stellt eine kleine Sammlung nützlicher Assembler-Routinen für die Erstellung von Maschinen-Code vor.

Tastatur-Abfrage

Die Tastatur-Abfrageroutine beginnt ab 2EF4H. Die Tastatur wird abgefragt und der Tastencode im Register A abgelegt. Die Register AF, BC, DE und HL werden geändert. Das Beispiel (12.1) wartet auf die Betätigung der Taste RETURN.

```
SCAN    CALL 2EF4H          ;TASTATURABFRAGE
        OR A                ;TASTE GEDRÜCKT?
        JR Z,SCAN           ;NICHT GEDRÜCKT
        CP ØDH              ;TEST CR
        JR NZ,SCAN.         ;KEIN CR
        RET                 ;ZURÜCK Z. AUFRUFENDEN PROG.
                                                   12.1
```

CRUN und CLOAD

Die Routine CRUN wird bei 372EH aufgerufen, CLOAD bei 3656H. HL wird als Zeiger auf den Programmnamen benutzt. Der Name wird in einem Buffer abgelegt. Die Anführungszeichen sind Bestandteile des Namens. Beendet wird der String mit einem '0'-Byte. Beide Routinen benutzen das System-RAM des LASER-Betriebssystems. Anwenderprogramme sollten diesen Bereich nicht benutzen.

```
NAME    DEFM '"TARGET"'     ;NAME DES ZU LADENDEN PROG.
        DEFB Ø              ;ABSCHLUSS
        ,
        ,
        LD HL,NAME          ;ZEIGER PROGR.NAME IN HL
        JP 372EH            ;CRUN AUSFÜHREN
                                                   12.2
```

```
GAME      DEFB Ø                  ;CLOAD OHNE PROGR.NAME
          ,
          ,
          LD HL,GAME              ;ZEIGER AUF NAME
          JP 3656H                ;CLOAD AUSFÜHREN
                                                    12.3
```

Character zum Bildschirm

Die Zeichen-Ausgaberoutine wird bei Q33AH aufgerufen. Das mit ASCII-Code in A definierte Zeichen wird auf die vom Cursor-Zeiger bestimmte Stelle des Bildschirmes ausgegeben. Register werden nicht geändert.

```
          LD A,'A'                ;CODE 'A' IN REGISTER A
          CALL 033AH              ;ZEICHEN AUSGEBEN
          LD A,0DH                ;CARRIAGE RETURN AUSGEBEN
          CALL 033AH
                                                    12.4
```

String-Ausgabe

Startadresse ist 28A7H. Das Registerpaar HL zeigt auf die Zeichenkette. Sie wird mit einem '0'-Byte abgeschlossen. Alle Register werden benutzt.

```
          LD HL,MSG               ;HL IST ZEIGER AUF STRING
          CALL 28A7H              ;STRING AUSGEBEN
          ,
          ,
MSG       DEFM 'READY'            ;STRING
          DEFB 0DH                ;CARRIAGE RETURN
          DEFB Ø                  ;TERMINATOR
                                                    12.5
```

RST 08H - Vergleiche Symbol

Eine Zeichenkette, auf die der Zeiger in HL zeigt, wird mit den Zeichencodes nach den RST 08H-Aufrufen verglichen. Wird Übereinstimmung festgestellt, so wird HL inkrementiert, die Kontrolle an die folgende RST 08H-Anweisung übergeben und das nächste Zeichen geprüft. Wird keine Übereinstimmung festgestellt, so wird eine Fehlermeldung (SYNTAX ERROR) ausgegeben und das Programm verzweigt zur erneuten Eingabe.

```
;
;VERGLEICHT ZEICHENKETTE, AUF DIE HL ZEIGT
;MIT DER KETTE 'A=B=C'
;
        RST 08H             ;TEST, OB 'A'
        DEFB 41H            ;HEXCODE FÜR A
        RST 08H             ;A GEFUNDEN
        DEFB 3DH            ;NUN TEST, OB '='
        RST 08H             ;OK, NUN TEST,
        DEFB 42H            ;OB 'B'
        RST 08H             ;OK, TEST
        DEFB 3DH            ;OB '='
        RST 08H             ;OK, TEST
        DEFB 43H            ;OK, STRING IST A=B=C
        ,
        ,
                                                12.6
```

Teste nächstes Symbol

A wird mit dem nächsten Zeichen geladen, auf das das Registerpaar HL zeigt. Das Carry-Flag wird gesetzt, wenn es ein alphanumerisches Zeichen ist. Leerzeichen und die Kontrollcodes 0BH und 09H werden ignoriert und das nächste Zeichen geladen und getestet. Die zu testende Zeichenkette muß mit '0' abgeschlossen sein. HL zeigt auf die Anfangsadresse der Zeichenkette minus 1 und wird vor dem Laden eines Zeichens jeweils mit 1 inkrementiert.

```
;
;EINE VON HL BEZEICHNETE ZEICHENKETTE WIRD GEPRÜFT, OB
;SIE TEIL EINER VARIABLEN-ZUWEISUNG IST, OB AUF DAS '='
;EINE KONSTANTE ODER EIN VARIABLENNAME FOLGT.
;
        RST 08H             ;TEST, OB '=' , JA?: FOLGENDES
        DEFB 3DH             ZEICHEN ADRESSIEREN
                            ;AUF '=' FOLGENDES ZEICHEN IN
NEXT    JR NC,VAR           ;NC, WENN VARIABLEN-NAME
        CALL 1E5AH          ;HOLE WERT DER KONSTANTE
        JR SKIP             ;NAECHSTER JOB
VAR     CP 2BH              ;NICHT NUMERISCH  +, -, ALPHA?
        JR Z,NEXT           ;+, NAECHSTES ZEICHEN
        CP 3DH              ;TEST OB -
        JR Z,NEXT           ;-, NAECHSTES ZEICHEN
        CALL 260DH          ;OK, ALPHAZEICHEN, JETZT NACH
                            ;VARIABLENNAMEN SUCHEN
        ,                                           12.7
```

RST 18 - Vergleiche DE mit HL

DE und HL werden numerisch miteinander verglichen. Es wird DE von HL abgezogen. Vorzeichenbehaftete Integer-Zahlen können nur im positiven Bereich bearbeitet werden. Es wird nur das A-Register benutzt. Das Ergebnis des Vergleiches wird im Status-Register abgelegt:

```
CARRY SET            - HL < DE
NO CARRY             - HL > DE
NZ                   - UNGLEICH
Z                    - GLEICH


;IN EINEM STRING (HL ZEIGT AUF ANFANG) WIRD DER NACH
;'=' FOLGENDE WERT GETESTET, OB ER IM BEREICH 100-500
;LIEGT.
;
        RST 08H             ;TEST AUF '='
        DEFB 3DH            ;'='
                            ;'=' GEFUNDEN, NAECHSTES ZEICHEN
        JR NZ,ERR           ;WENN NICHT NUMERISCH
        CALL 1E5AH          ;HOLE BINAERWERT
        LD HL,500           ;WERT OBERGRENZE
        RST 18H             ;VERGLEICH MIT BINWERT
        JR C,ERR            ;CARRY, WENN WERT >500
        LD HL,100           ;TEST UNTERE GRENZE
        RST 18H             ;VERGLEICH
        JR NZ,ERR           ;KEIN CARRY, FEHLER!
        ,
                                                        12.8
```

RST 20H - Test Variablen-Typ

Die Routine kehrt mit einer Kombination numerischer Werte im A-Register und Statusflags zurück. Sie orientiert sich am 'Data Mode Flag' (78AFH) und legt so den Zahlentyp im Rechenregister WRA1 fest. Der Einsatz sollte kontrolliert erfolgen, da einige aufgerufene Unterprogramme das Typflag ändern können und so der Wert in WRA1 und das Flag nicht mehr übereinstimmen.

```
    TYP       STATUS      A-REG
------------------------------
02  INTEGER   NZ/C/M/E     -1
03  STRING    Z/C/P/E       0
04  SNG.PREC  NZ/C/P/O      1
08  DBL.PREC  NZ/NC/P/E     5


;NACH EINER INTEGER-ADDITION WIRD DER DATENTYP
;GETESTET, UM BEI OVERFLOW DEN TYP ZU BESTIMMEN
;
        LD A,2              ;TYPCODE INTEGER
        LD (78AFH),A        ;IN FLAGREGISTER
        LD BC,(VAL1)        ;ERSTER WERT
        LD HL,(VAL2)        ;ZWEITER WERT
        CALL 0B2DH          ;DO INTEGER ADDITION
        RST 20H             ;TEST AUF OVERFLOW
        JP M,OK             ;RESULTAT IST INTEGER
        ,                   ;NICHT INTEGER
        ,                   ;TESTE ANDERE TYPEN
OK      LD (SUM),HL         ;INTEGER ERGEBNIS ABLEGEN
        ,
VAL1    DEFW 125            ;16 BIT INTEGERZAHL
VAL2    DEFW 4235           ;16 BIT INTEGERZAHL
SUM     DEFW Ø              ;HIER KOMMT ERGEBNIS HIN
                                                  12.9
```

Bildschirm-Control-Codes

Alle Cursor-Funktionen können in Maschinenprogrammen ausgeführt werden.

```
CURSOR LINKS    08H
CURSOR AUFW.    1BH
CURSOR ABW.     0AH
CURSOR RECHTS   09H
RUBOUT          7FH
INSERT          15H
CURSOR HOME     1CH
CLEAR SCREEN    1FH


        LD A,1CH            ;CURSOR HOME
        CALL 033AH          ;DO
        LD A,1FH            ;CLEAR SCREEN
        CALL 033AH          ;DO
                                 12.10
```

Programmieren der Joysticks

In Programmen mit Maschinencode können die Joysticks noch einfacher als unter BASIC gelesen werden. Das folgende Beispiel liest die Joystick-Matrix und liefert als Resultat in den Registern B und C den Joystick-Status. B enthält den Status des rechten Joysticks, C den des linken. (Siehe auch Kapitel 6: Programmieren der Joysticks.)

```
BIT   5        4        3        2        1       0
      FIRE A   FIRE B   LINKS    RECHTS   ABW.    AUFW.

JOYSTK   IN A,(2EH)           ;1. REIHE LESEN
         OR 0E0H
         CPL
         LD B,A
         IN A,(2DH)           ;2. REIHE
         BIT 4,A
         JR NZ,JOYST1
         SET 5,B              ;RECHTER JOYSTICK STATUS IN B
JOYST1   IN A,2BH             ;3. REIHE LESEN
         OR 0E0H
         CPL
         LD C,A
         IN A,(27H)           ;4. REIHE LESEN
         BIT 4,A
         RET NZ
         SET 5,C              ;STATUS LINKER JOYSTICK IN C
         RET
                                                      12.11
```

Buzzer

Unter der Adresse 345CH ist die Routine zu finden, die den Piezo-Lautsprecher steuert. Vor Aufruf muß das Register HL mit einer Zahl, entsprechend der gewünschten Tonfrequenz (Pitch), geladen werden und das BC-Register mit dem Code der Tondauer (Duration). Alle Register werden von der Routine benutzt.

```
LD HL,259           ;FREQUENZCODE
LD BC,75            ;TONDAUER
CALL 345CH          ;TONROUTINE AUFRUFEN
                                   12.12

CALL 3450H          ;'BEEP' ERZEUGEN
                                   12.13
```

Die Codierung der Tonfrequenz ist umgekehrt proportional zur Tonfrequenz. Kleine Zahlen repräsentieren hohe Frequenzen und große Zahlen tiefe. Beispielsweise wird das 'kleine C' mit der Dezimalzahl 526, das 'C' mit der Zahl 259 und das 'hohe C' mit 127 codiert. Die folgende Routine (10.12) erzeugt einen Ton 'C' mit einer Tondauer von 75 Schwingungen.

Beep-Routine

Die vom Betriebssystem verwendete Routine, welche bei jeder Tastenbetätigung den charakteristischen 'Beep' erzeugt, beginnt ab Adresse 3450H. Bis auf das HL-Register werden alle Registerinhalte zerstört. Um einen solchen Ton zu produzieren, wird einfach der Aufruf gemäß (12.13) gegeben.

Drucker-Steuerung

Um ein Zeichen an den Drucker auszugeben, ist der Zeichencode in das C-Register zu laden und die Drucker-Treiberroutine in 058DH aufzurufen. Nach dem Aufruf ist der ASCII-Code des Zeichens im A- und im C-Register zur weiteren Bearbeitung enthalten, der Inhalt aller anderen Register ist zerstört. Der Buchstabe 'a' (ASCII 97 dezimal) wird so ausgegeben (12.14):

```
LD C,97              ;CODE IN C-REG LADEN
CALL 058DH           ;DRUCKER-TREIBER AUFRUFEN
                                        12.14
```

Wird ein CARRIAGE RETURN-Steuercode an den Drucker über den Treiber gesendet, so wird automatisch ein LINE FEED-Code nachgesendet. Wird der Treiber mit C-Register gleich '0' aufgerufen, so prüft die Treiber-Routine den Drucker-Status und liefert das Ergebnis mit Bit 0 des A-Registers gesetzt oder gelöscht. Außerdem wird die BREAK-Taste getestet und bei gedrückter Taste das Carry-Flag gesetzt.

Drucker-Status

Eine Routine, die den Drucker-Status prüft, beginnt ab 05C4H. Nach Aufruf lädt sie den Status (I/O-Port 00H) in das A-Register und kehrt zurück. Bit 0 ist '1', wenn vom Drucker das Signal 'BUSY' erkannt wird, und '0', wenn der Drucker empfangsbereit ist. Weitere Register werden nicht verändert.

```
TEST    CALL 05C4H          ;TEST DRUCKER READY
        BIT Ø,A             ;TEST BIT Ø
        JR NZ,TEST          :WARTEN, WENN BUSY
                                             12.15
```

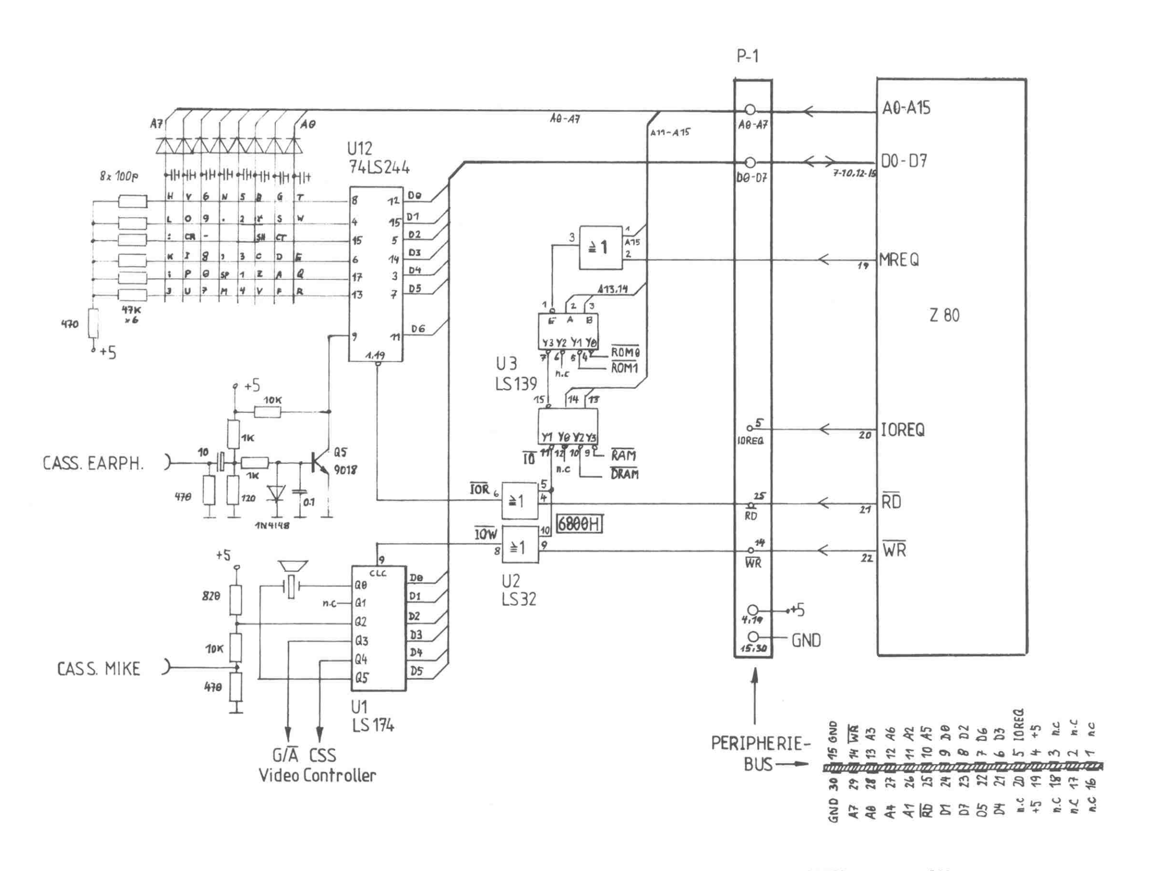
CR/LF an Drucker

Um die CR/LF-Kombination an den Drucker auszugeben, kann eine ab 3AE2H beginnende Routine aufgerufen werden. Es brauchen keine Register vorher gesetzt zu werden. Die Inhalte aller Register werden geändert. Wird während des Druckvorganges oder während der READY-Status abgewartet wird, die BREAK-Taste betätigt, kehrt die Routine mit gesetztem Carry-Flag zurück.

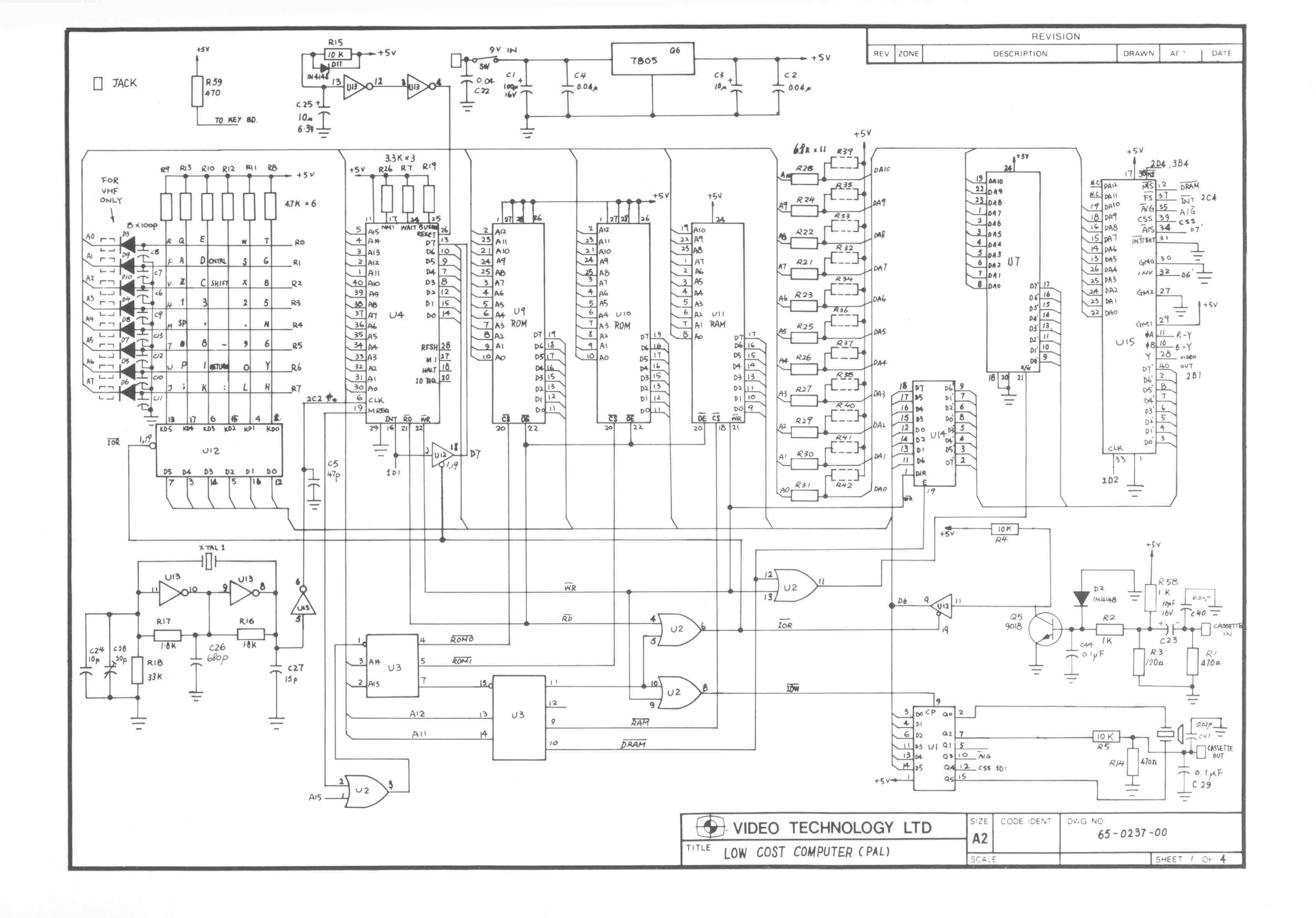
```
CALL 3AEH                ;SENDET CR / LF AN DEN DRUCKER
JP C,BRK                 ;NACH 'BRK', WENN TASTE GEDRUECKT
...                      ;WEITER, WENN NICHT
,
                                            12.16
```

Anhänge

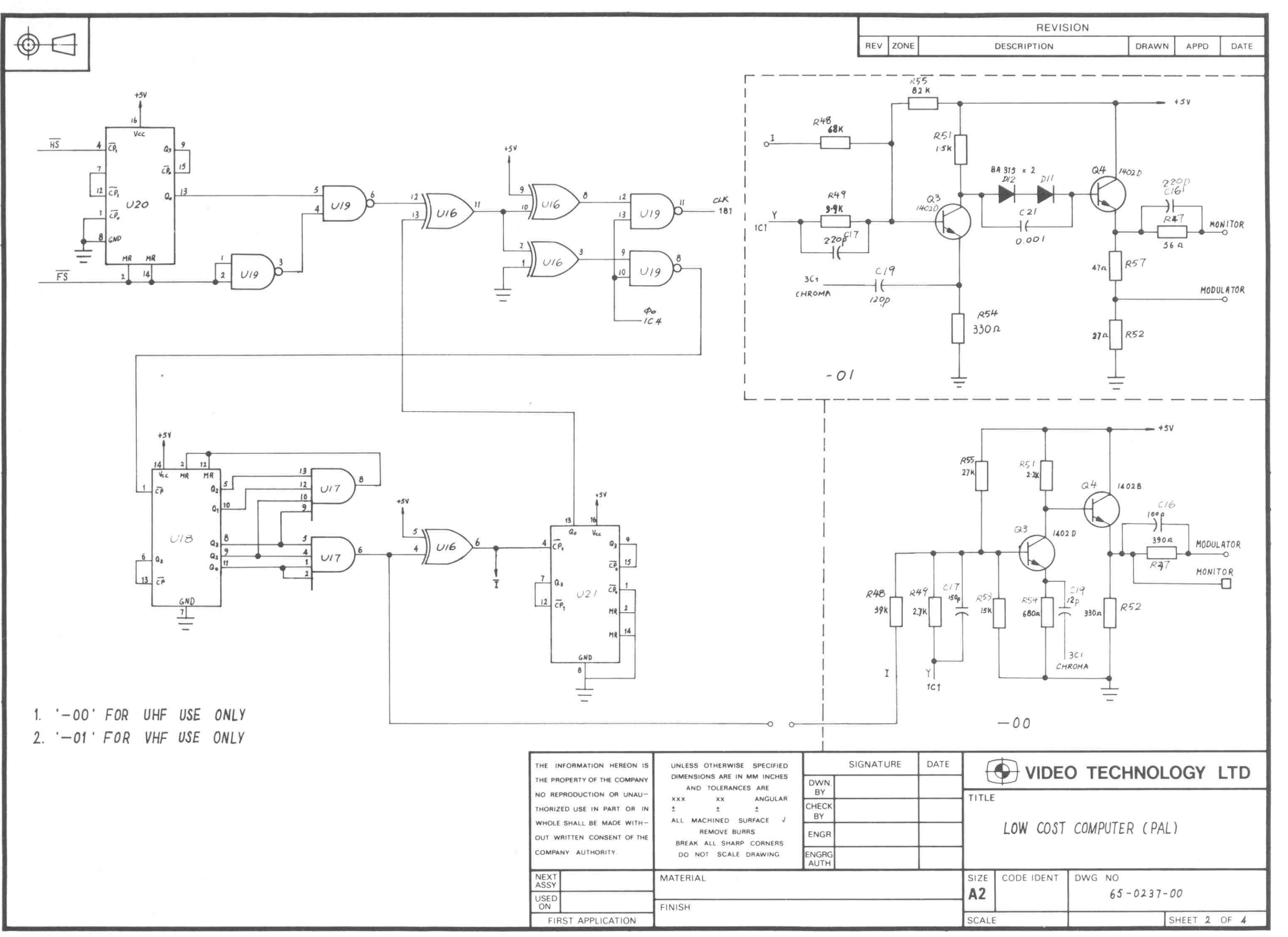
Ein Inhaltsverzeichnis für die Anhänge befindet sich am Buchbeginn.



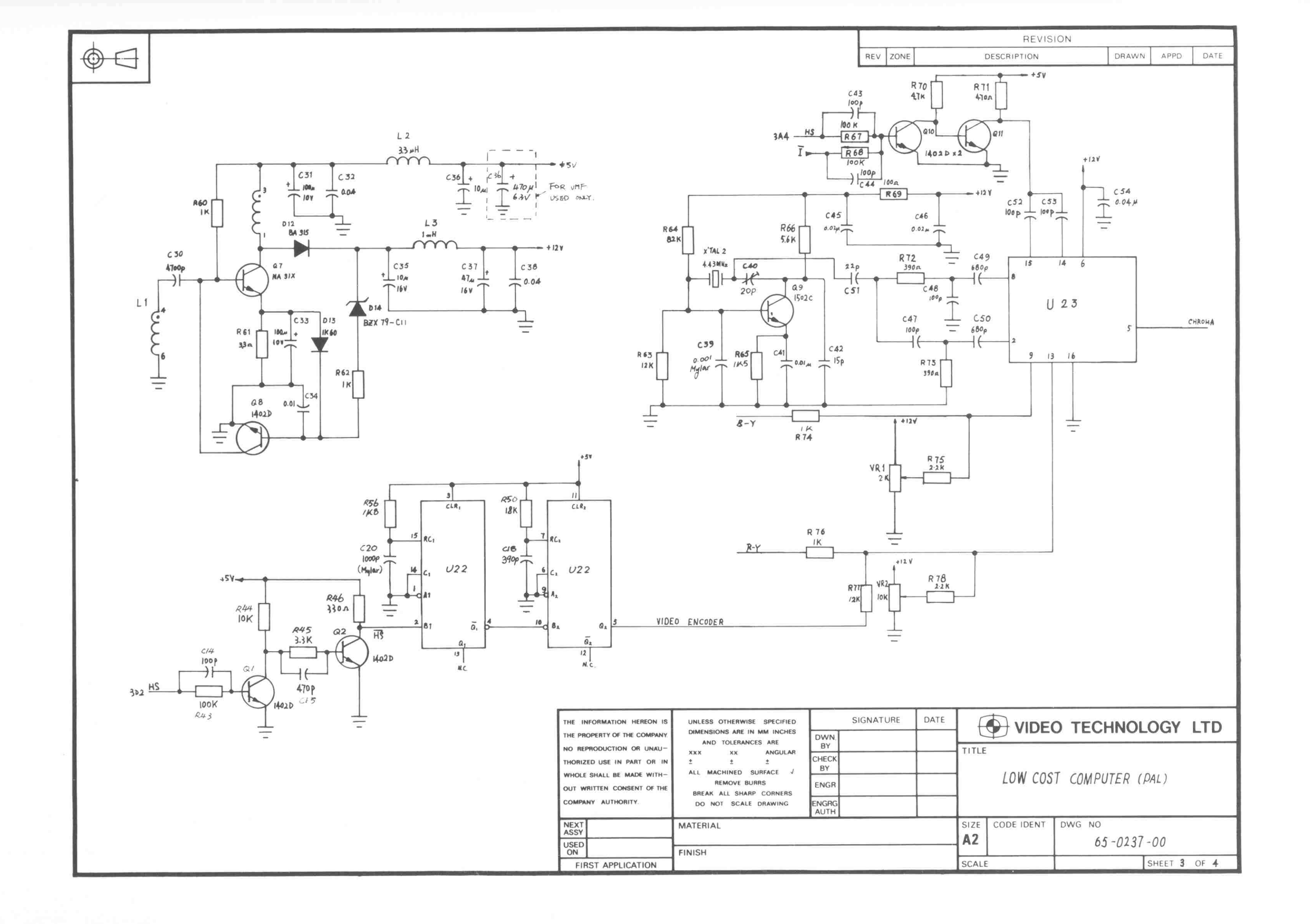
Z 80
A0-A15
D0-D7
MREQ
IOREQ
RD
WR
P-1
U12
74LS244
U3
LS139
U2
LS32
U1
LS174
ROM0
ROM1
RAM
DRAM
IOR
IOW
6800H
Q5
9018
1N4148
CASS. EARPH.
CASS. MIKE
G/A CSS
Video Controller
8x 100P
47K x6
PERIPHERIE-
BUS
GND
+5
LASER 110/210: I/O



REVISION
REV
ZONE
DESCRIPTION
DRAWN
DATE
JACK
R59 470
TO KEY BD.
R15 10 K
D11 IN4148
C25 10µ 6.3V
9V IN
SW
C22 0.04
C1 100µ 16V
C4 0.04µ
7805
Q6
C3 10µ
C2 0.04µ
FOR VHF ONLY
8 × 100p
47K × 6
3.3K × 3
U4
U9 ROM
U10 ROM
U11 RAM
6.8K × 11
U7
U15
U14
U12
U13
U3
U2
U1
XTAL 1
R17 1.8K
R16 1.8K
R18 33K
C24 10p
C28 20p
C26 680p
C27 15p
C5 47p
Q5 9018
D2 IN4148
R58 1K
R2 1K
R3 120Ω
R1 470Ω
C44 0.1µF
C23 10µF 16V
C40
CASSETTE IN
R4 10K
R5 10K
R14 470Ω
C41
C29 0.1µF
CASSETTE OUT
VIDEO OUT
VIDEO TECHNOLOGY LTD
TITLE LOW COST COMPUTER (PAL)
SIZE A2
CODE IDENT
DWG NO 65-0237-00
SCALE
SHEET 1 OF 4



REVISION
REV
ZONE
DESCRIPTION
DRAWN
APPD
DATE
U20
U19
U16
U18
U17
U21
CLK
181
MONITOR
MODULATOR
CHROMA
-01
-00
1. '-00' FOR UHF USE ONLY
2. '-01' FOR VHF USE ONLY
THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY. NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY.
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
ALL MACHINED SURFACE
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING
SIGNATURE
DATE
DWN. BY
CHECK BY
ENGR
ENGRG AUTH
VIDEO TECHNOLOGY LTD
TITLE
LOW COST COMPUTER (PAL)
NEXT ASSY
USED ON
FIRST APPLICATION
MATERIAL
FINISH
SIZE
A2
CODE IDENT
DWG NO
65-0237-00
SCALE
SHEET 2 OF 4



THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY. NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY.

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
XXX ± XX ± ANGULAR ±
ALL MACHINED SURFACE √
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING

	SIGNATURE	DATE
DWN. BY		
CHECK BY		
ENGR		
ENGRG AUTH		

VIDEO TECHNOLOGY LTD

TITLE: LOW COST COMPUTER (PAL)

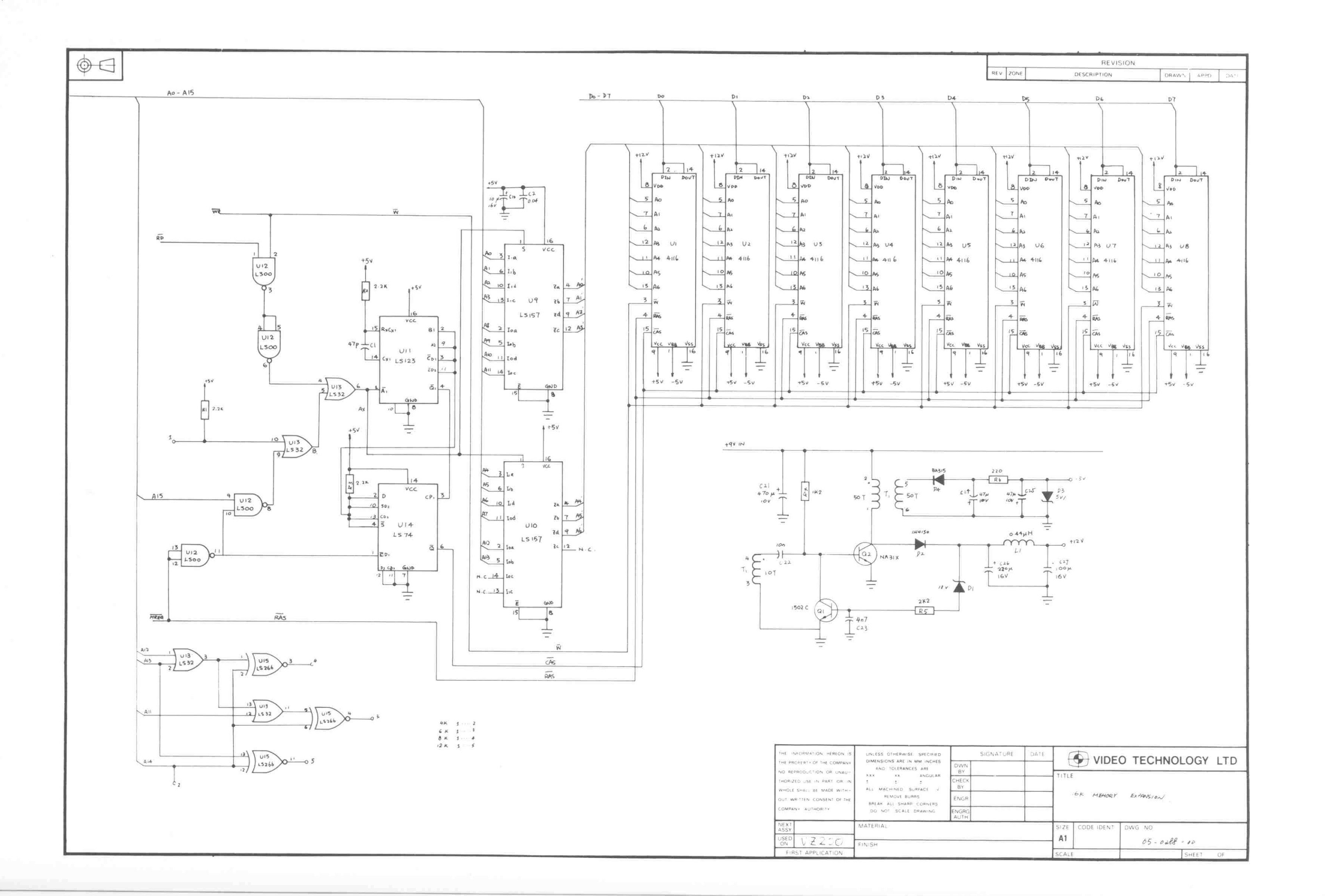
NEXT ASSY | USED ON | FIRST APPLICATION

MATERIAL

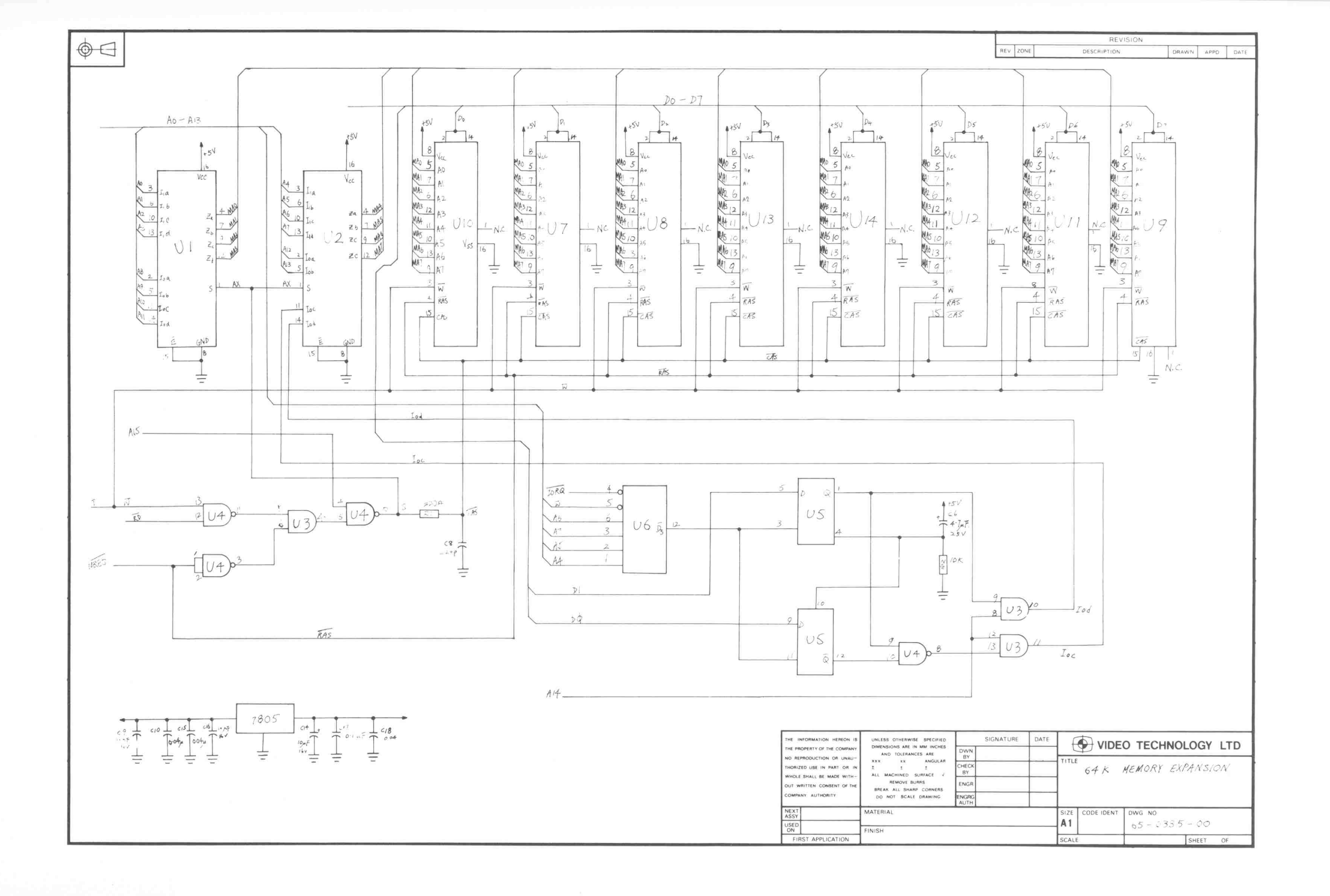
FINISH

SIZE: A2 | CODE IDENT | DWG NO: 65-0237-00

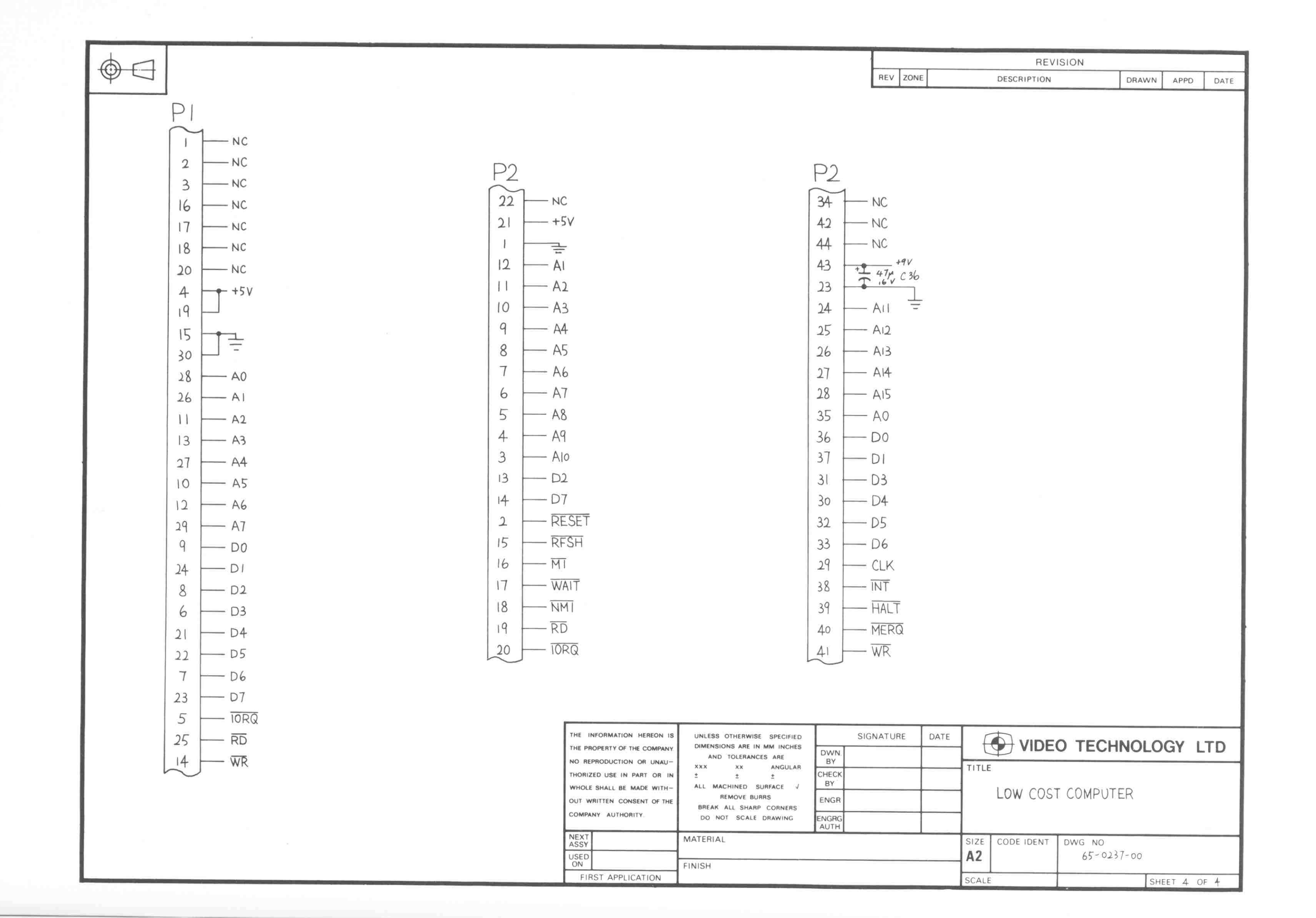
SCALE | SHEET 3 OF 4

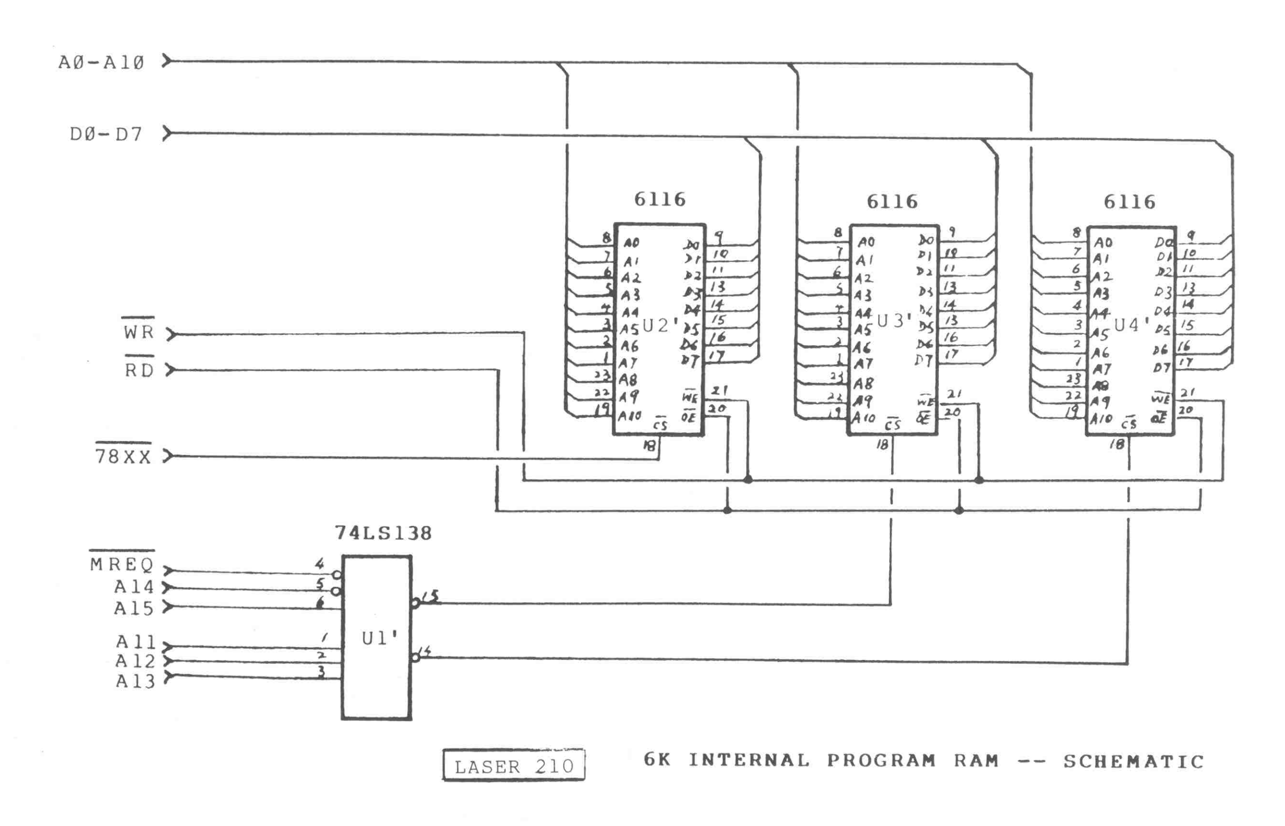


REVISION
REV ZONE DESCRIPTION DRAWN APPD DATE
THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY. NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY.
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
ALL MACHINED SURFACE
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING
SIGNATURE DATE
DWN BY
CHECK BY
ENGR
ENGRG AUTH
VIDEO TECHNOLOGY LTD
TITLE
16K MEMORY EXPANSION
NEXT ASSY
USED ON VZ200
FIRST APPLICATION
MATERIAL
FINISH
SIZE A1
CODE IDENT
DWG NO 05-0288-10
SCALE
SHEET OF



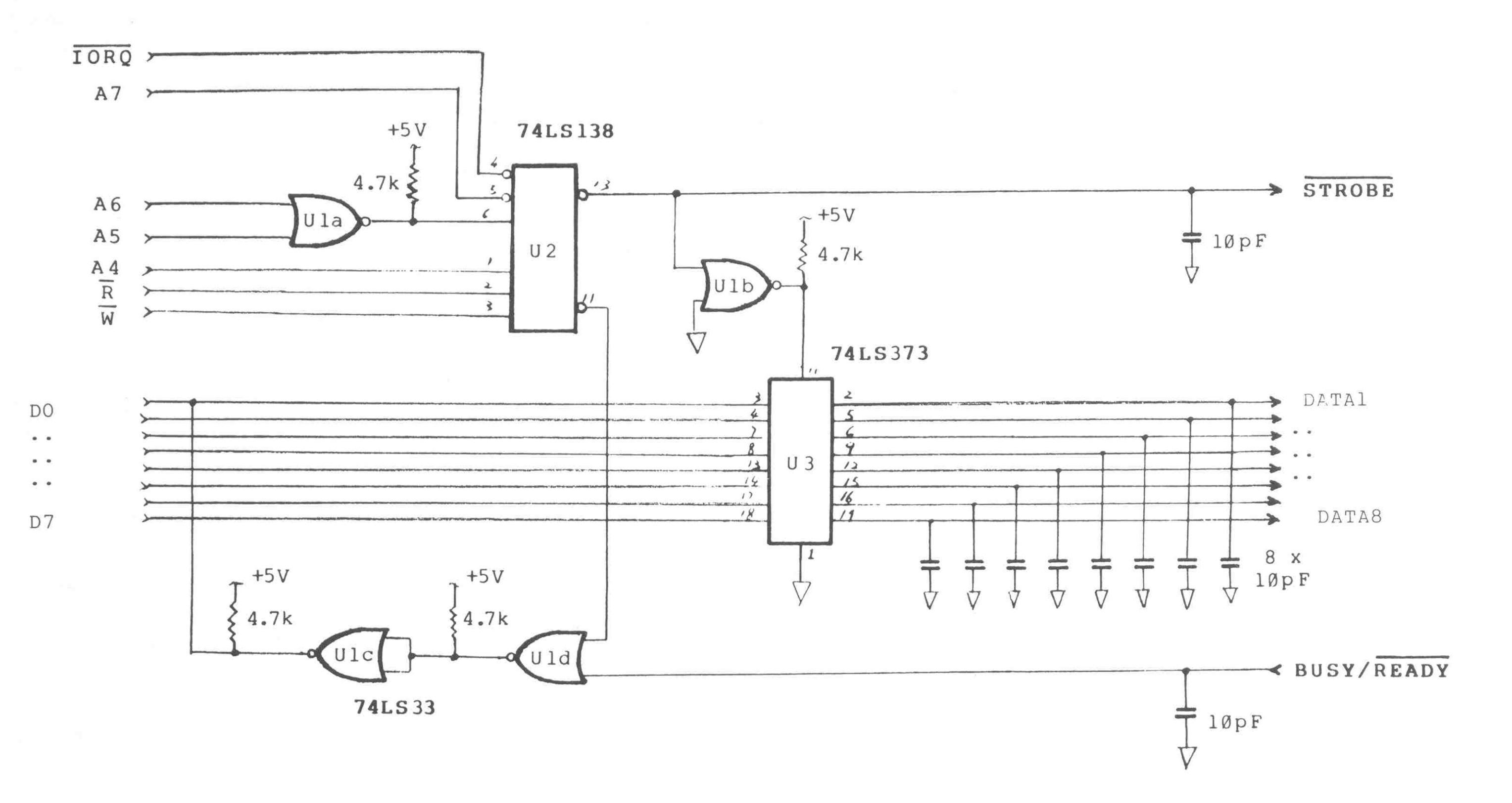
REVISION
REV
ZONE
DESCRIPTION
DRAWN
APPD
DATE
A0 – A13
D0 – D7
U1
U2
U10
U7
U8
U13
U14
U12
U11
U9
U6
U5
U4
U3
7805
A15
A14
IORQ
RD
MREQ
RAS
CAS
W
N.C.
+5V
10K
THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
ALL MACHINED SURFACE
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING
SIGNATURE
DATE
DWN BY
CHECK BY
ENGR
ENGRG AUTH
NEXT ASSY
USED ON
FIRST APPLICATION
MATERIAL
FINISH
VIDEO TECHNOLOGY LTD
TITLE
64 K MEMORY EXPANSION
SIZE
A1
CODE IDENT
DWG NO
65 - 0335 - 00
SCALE
SHEET
OF


REVISION
REV
ZONE
DESCRIPTION
DRAWN
APPD
DATE
P1
1 NC
2 NC
3 NC
16 NC
17 NC
18 NC
20 NC
4 +5V
19
15
30
28 A0
26 A1
11 A2
13 A3
27 A4
10 A5
12 A6
29 A7
9 D0
24 D1
8 D2
6 D3
21 D4
22 D5
7 D6
23 D7
5 IORQ
25 RD
14 WR
P2
22 NC
21 +5V
1
12 A1
11 A2
10 A3
9 A4
8 A5
7 A6
6 A7
5 A8
4 A9
3 A10
13 D2
14 D7
2 RESET
15 RFSH
16 M1
17 WAIT
18 NMI
19 RD
20 IORQ
P2
34 NC
42 NC
44 NC
43 +9V
47μ 16V C36
23
24 A11
25 A12
26 A13
27 A14
28 A15
35 A0
36 D0
37 D1
31 D3
30 D4
32 D5
33 D6
29 CLK
38 INT
39 HALT
40 MERQ
41 WR
THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY.
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
XXX ± XX ± ANGULAR ±
ALL MACHINED SURFACE √
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING
SIGNATURE
DATE
DWN BY
CHECK BY
ENGR
ENGRG AUTH
VIDEO TECHNOLOGY LTD
TITLE
LOW COST COMPUTER
NEXT ASSY
USED ON
FIRST APPLICATION
MATERIAL
FINISH
SIZE
A2
CODE IDENT
DWG NO
65-0237-00
SCALE
SHEET 4 OF 4

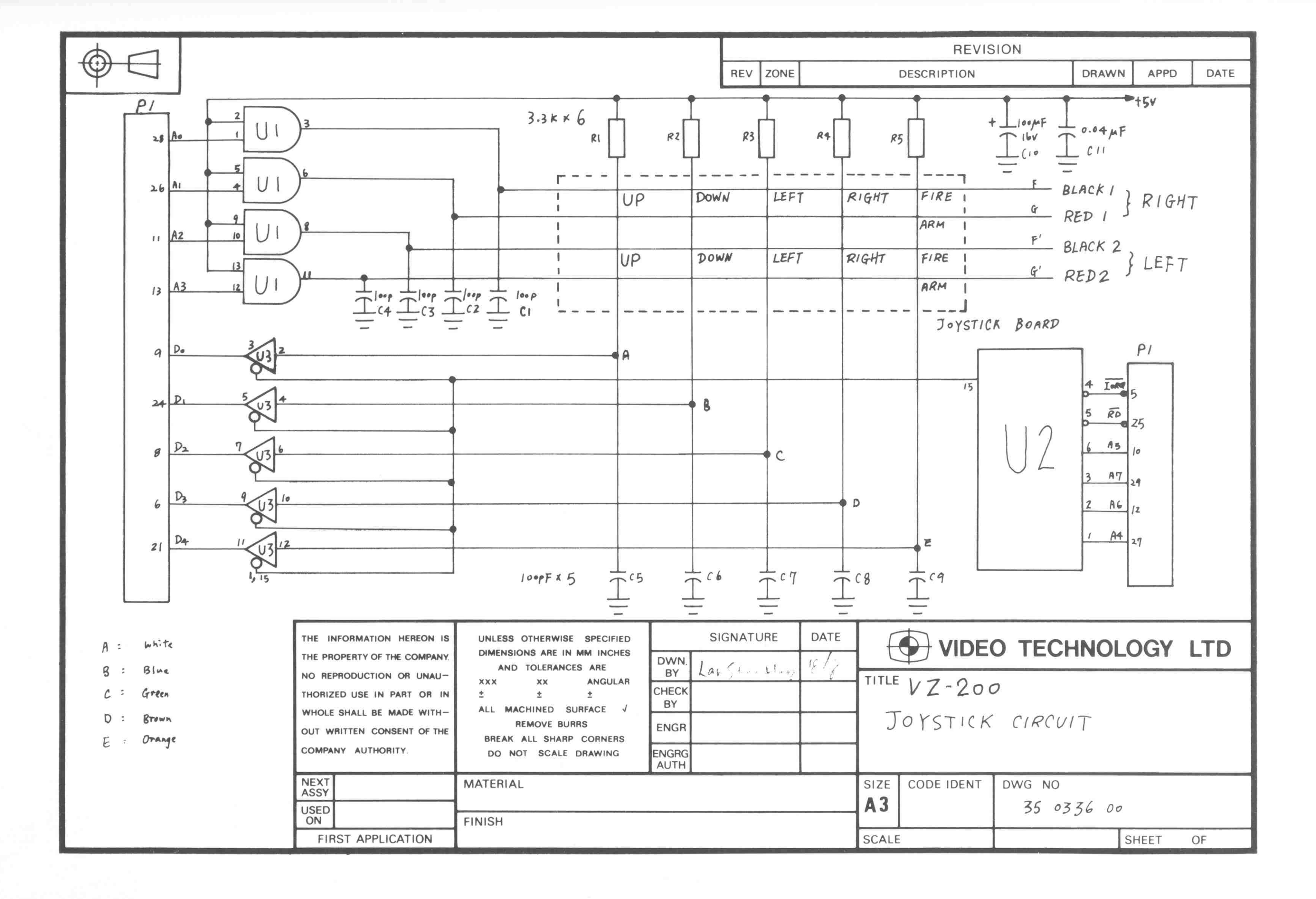


LASER 210

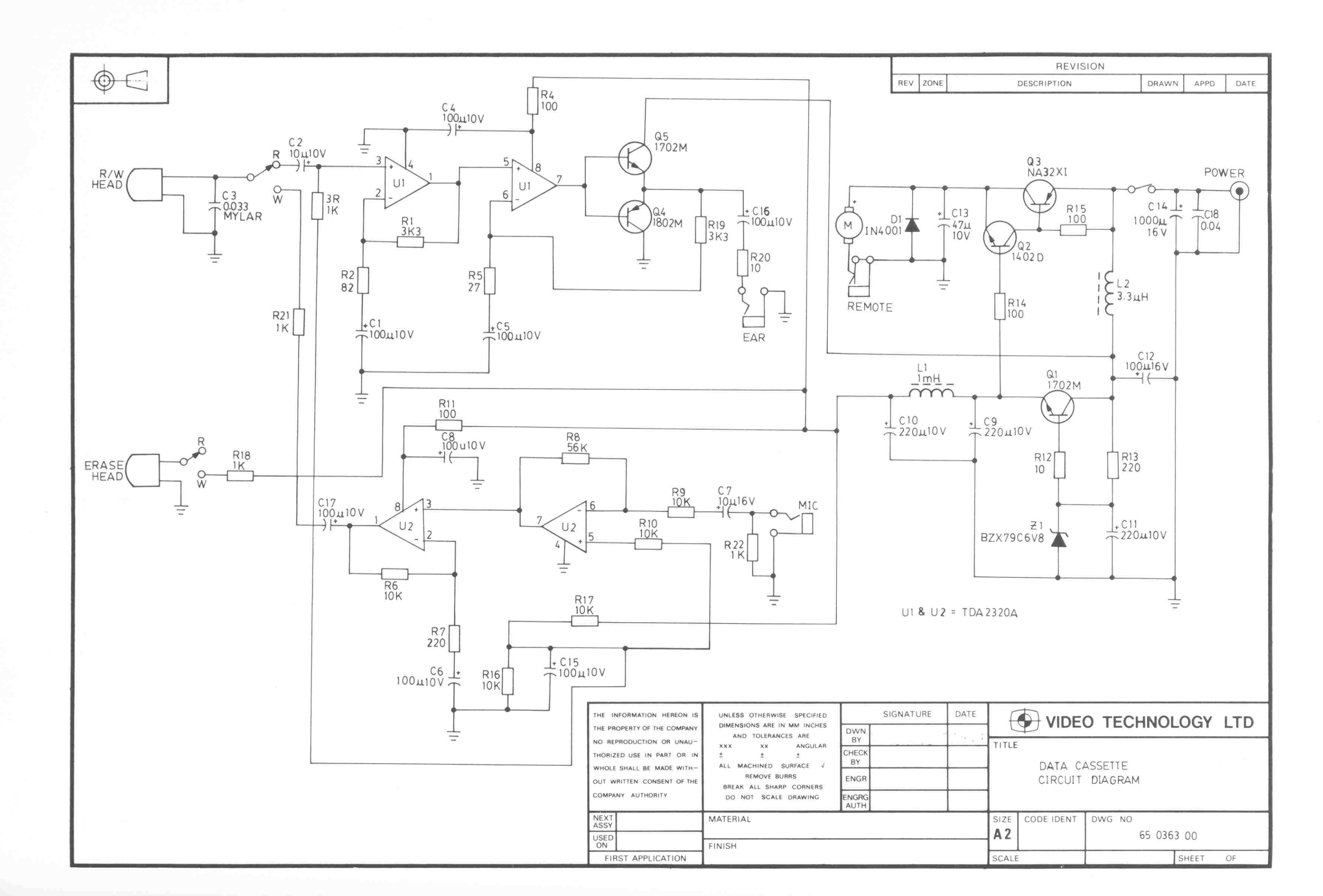
6K INTERNAL PROGRAM RAM -- SCHEMATIC

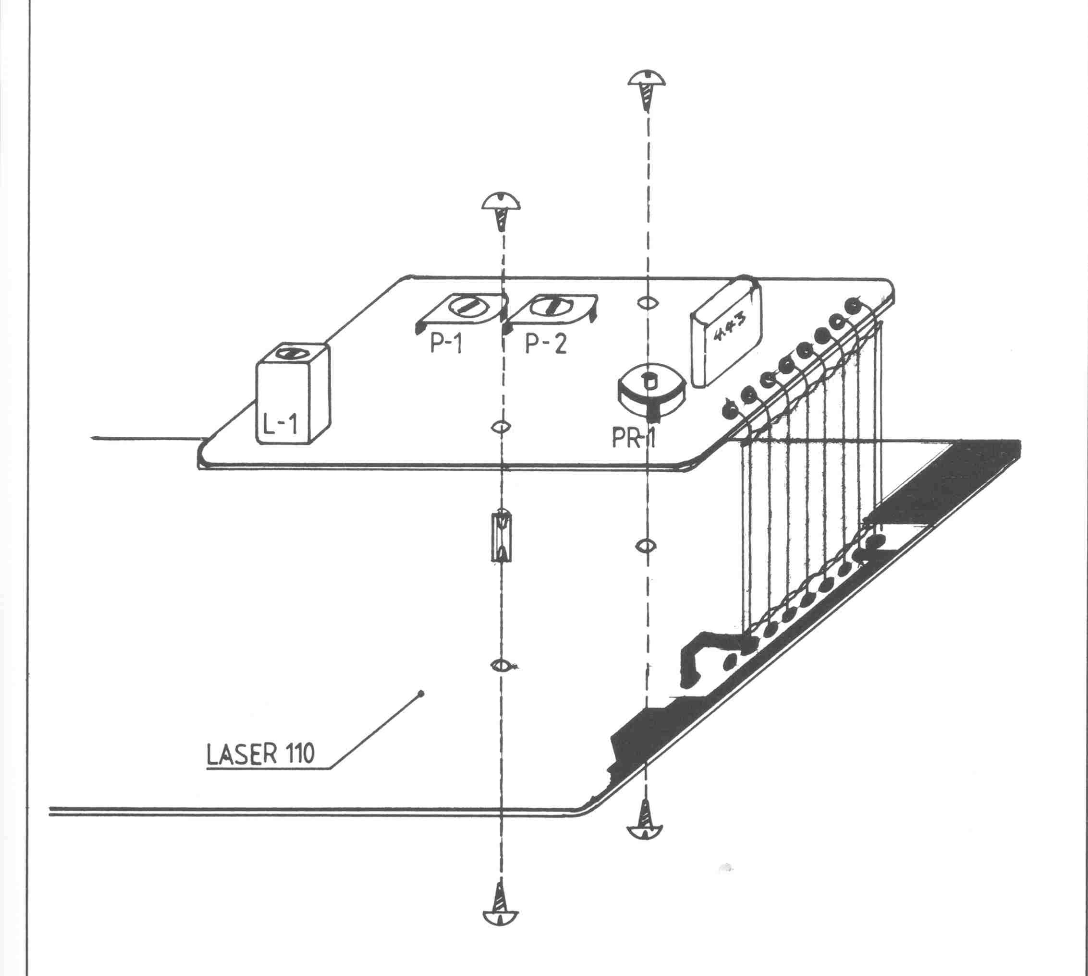


PRINTER INTERFACE -- SCHEMATIC



REVISION
REV
ZONE
DESCRIPTION
DRAWN
APPD
DATE
P1
3.3K x 6
R1
R2
R3
R4
R5
+5V
100μF
16V
C10
0.04μF
C11
U1
U2
U3
A0
A1
A2
A3
D0
D1
D2
D3
D4
UP
DOWN
LEFT
RIGHT
FIRE
ARM
BLACK 1
RED 1
RIGHT
BLACK 2
RED2
LEFT
JOYSTICK BOARD
100p
C1
C2
C3
C4
IORQ
RD
A5
A7
A6
A4
100pF x 5
C5
C6
C7
C8
C9
A : White
B : Blue
C : Green
D : Brown
E : Orange
THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY. NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY.
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
XXX
XX
ANGULAR
±
±
±
ALL MACHINED SURFACE √
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING
SIGNATURE
DATE
DWN. BY
CHECK BY
ENGR
ENGRG AUTH
VIDEO TECHNOLOGY LTD
TITLE VZ-200
JOYSTICK CIRCUIT
NEXT ASSY
USED ON
FIRST APPLICATION
MATERIAL
FINISH
SIZE
A3
CODE IDENT
DWG NO
35 0336 00
SCALE
SHEET
OF


REVISION
REV
ZONE
DESCRIPTION
DRAWN
APPD
DATE
R/W HEAD
C3 0.033 MYLAR
R
W
C2 10µ10V
3R 1K
R21 1K
C4 100µ10V
R4 100
U1
R1 3K3
R2 82
C1 100µ10V
R5 27
C5 100µ10V
Q5 1702M
Q4 1802M
R19 3K3
C16 100µ10V
R20 10
EAR
ERASE HEAD
R18 1K
R11 100
C8 100µ10V
R8 56K
C17 100µ10V
U2
R6 10K
R7 220
C6 100µ10V
R16 10K
C15 100µ10V
R17 10K
R9 10K
C7 10µ16V
R10 10K
R22 1K
MIC
M
D1 IN4001
C13 47µ 10V
REMOTE
Q3 NA32XI
Q2 1402D
R15 100
R14 100
POWER
C14 1000µ 16V
C18 0.04
L2 3.3µH
C12 100µ16V
L1 1mH
Q1 1702M
C10 220µ10V
C9 220µ10V
R12 10
R13 220
Z1 BZX79C6V8
C11 220µ10V
U1 & U2 = TDA2320A
THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY. NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
XXX ± XX ± ANGULAR ±
ALL MACHINED SURFACE √
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING
SIGNATURE
DATE
DWN BY
CHECK BY
ENGR
ENGRG AUTH
VIDEO TECHNOLOGY LTD
TITLE
DATA CASSETTE
CIRCUIT DIAGRAM
NEXT ASSY
USED ON
FIRST APPLICATION
MATERIAL
FINISH
SIZE A2
CODE IDENT
DWG NO
65 0363 00
SCALE
SHEET
OF



LASER PAL COLOR BOARD

80 - PAL

EINBAU-ANLEITUNG

Abgleich

P-1, P-2
abwechselnd auf grünes Innenfeld und farblosen Rand einstellen
PR-1 auf Minimum-Randstörungen einstellen.
L-1 auf Minimum-Störungen im Bild einstellen.

Anhang 9: Variablen-Formate

```
10 POKE 30978,8 : REM VARIABLEN B IN DBL PREC
20 DIM A%, AA, BB, Z1$, AA (3,3,3)
30 Z1$ = "TEST"
40 AD = PEEK (30970) * PEEK (30969)
45 REM ZEIGER AUF VARIABLEN-TABELLE
50 FOR I = 0 TO 80
60 PRINT PEEK (AD + I);: NEXT I
```

A% INTEGER	2	0 65	0 0	WERT

SINGLE PRECISION AA	4	65 65	0 0 0 0

DOUBLE PRECISION BB	8	66 66	0 0 0 0 0 0 0 0

STRING Z1$	3	49 90	4	22 123

VARIABLEN-TYP (→ 3)
NAME (ASCII) (→ 49 90)
STRING-LAENGE (→ 4)
ZEIGER AUF TEXT-STRING (→ 22 123)

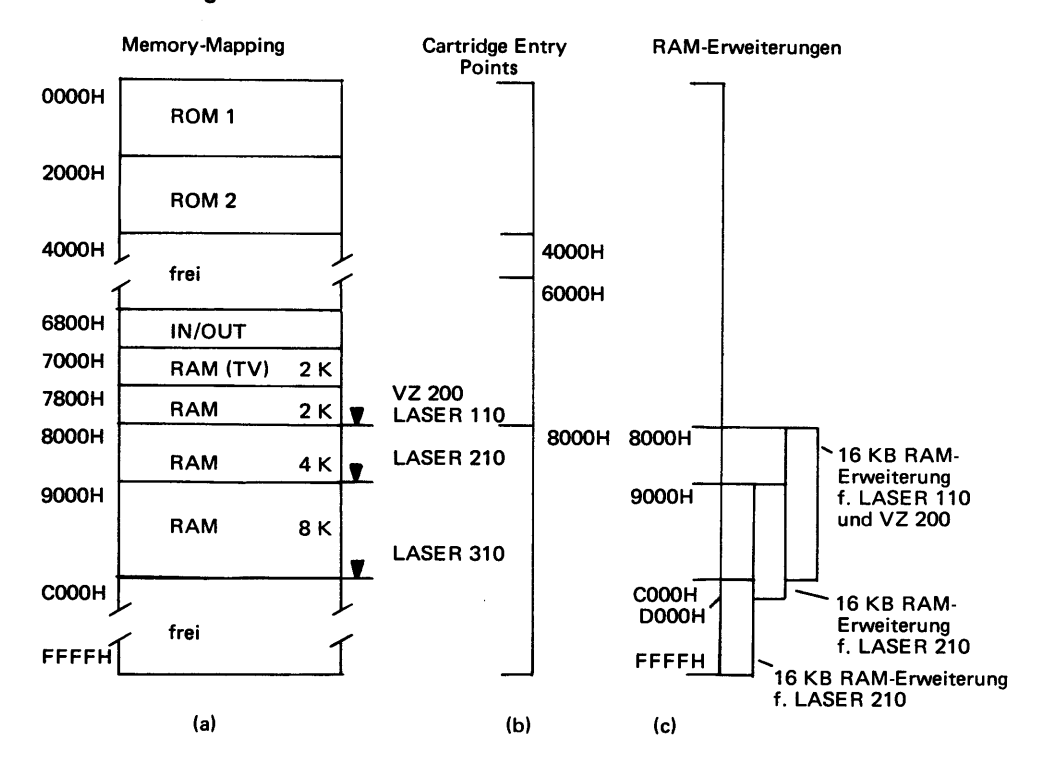
FOLGEN VARIABLEN AD, I

FELD AA (3,3,3)	4	65 65	7 1	3	4 0	4 0	4 0

ZAHL FOLGENDER BYTES (→ 7 1)
ANZAHL DIMENSION (→ 3)
ELEM. 1. / ELEM. 2. / ELEM. 3. DIMENSION

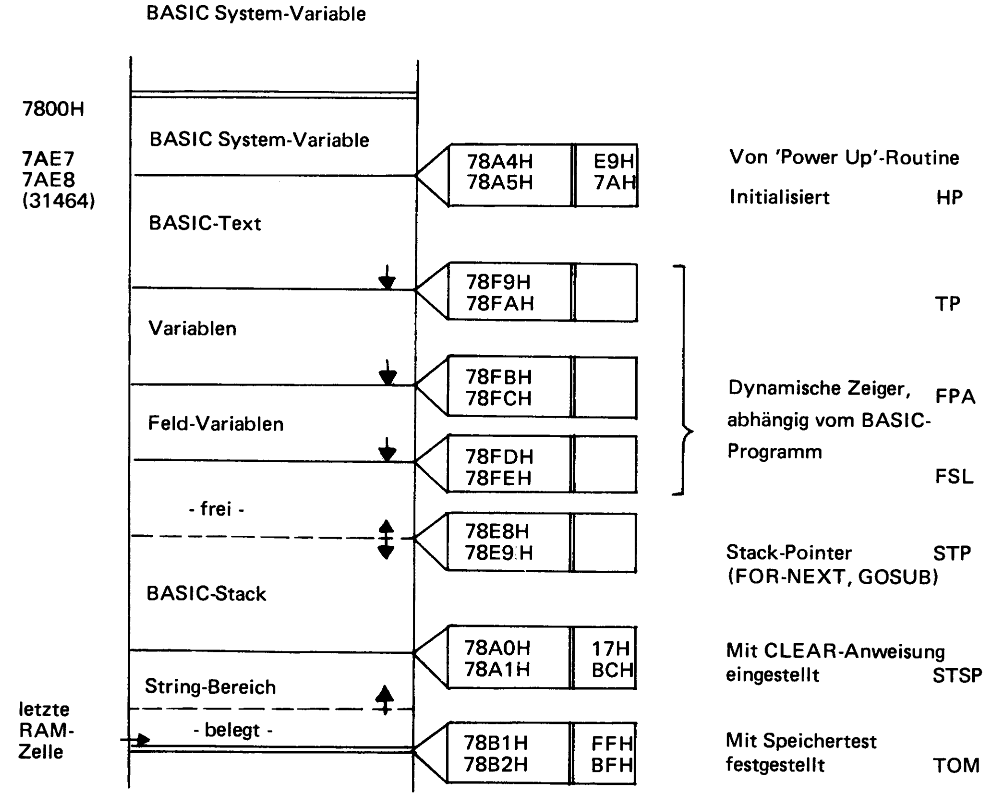
WERT .... 64 X 4 BYTES (GESAMTE FELDLAENGE=260 BYTES

Anhang 10



Memory-Mapping
Cartridge Entry Points
RAM-Erweiterungen
0000H
ROM 1
2000H
ROM 2
4000H
frei
6800H
IN/OUT
7000H
RAM (TV) 2 K
7800H
RAM 2 K
8000H
RAM 4 K
9000H
RAM 8 K
C000H
frei
FFFFH
VZ 200
LASER 110
LASER 210
LASER 310
4000H
6000H
8000H
8000H
9000H
C000H
D000H
FFFFH
16 KB RAM-
Erweiterung
f. LASER 110
und VZ 200
16 KB RAM-
Erweiterung
f. LASER 210
16 KB RAM-Erweiterung
f. LASER 210
(a)
(b)
(c)

Anhang 11



Zeigerwerte gelten für den LASER 110 mit 16 KB Speichererweiterung.
Die Pfeile markieren eine dynamische Grenze
Der Stringbereich ist mit CLEAR 1000 definiert

HP = Text Head Pointer
TP = Text Tail Pointer
FPA = Field Pointer Anfang
FSL = Free Space List
STP = BASIC Stack Pointer
STSP = String Space
TOM = Top of Memory

Anhang 12: System-Variablen

```
78ØC 3Ø732  nicht benutzt bis .....
7815 3Ø741
7816 3Ø742  ADRESSE DER KEYBOARD-SCAN-ROUTINE
7818 3Ø744  nicht benutzt bis .....
781C 3Ø748
782Ø 3Ø752  ZEIGER AUF CURSOR-ADRESSE
783B 3Ø779  COPIE DES OUTPUT-LATCH INHALTES
787D 3Ø845  INTERRUPT EXIT ADRESSE
            (KANN AUF EIGENE INTERR.ROUTINE GERICHTET WERDEN
788E 3Ø862  USR- SPRUNGVEKTOR
789Ø 3Ø864  ZUFALLSZAHLEN-GENERATOR
7893 3Ø867  'IN', ADRESSE, WERT
7896 3Ø87Ø  'OUT', ADRESSE, WERT
7999 3Ø873  LETZTES ZEICHEN NACH 'BREAK'.
789A 3Ø874  FLAG: EINTRITT IN RESUME-ROUTINE
789B 3Ø875  ANZ. ZEICHEN IN AKTUELLER 'PRINT'-ZEILE
789C 3Ø876  AUSGABE-KANAL: 1=PRINTER, Ø=VIDEO, -1=KASSETTE
789D 3Ø877  FORMAT DER DISPLAY-ZEILE
789E 3Ø878  FORMAT DER PRINT-ZEILE
789F 3Ø979  reserviert

78AØ 3Ø88Ø  ZEIGER AUF STRING-BEREICH
78A2 3Ø882  AKTUELLE ZEILENNUMMER
78A4 3Ø884  ADRESSE BASIC START
78A6 3Ø886  CURSOR COLUMN POSITION
78A7 3Ø887  ADRESSE TASTATUR BUFFER
78A9 3Ø889  FLAG: Ø=KASSETTE INPUT, SONST <>Ø
78AA 3Ø89Ø  ZUFALLSZAHLENGENERATOR
78AB 3Ø891  WERT DES REFRESH-REGISTERS
78AC 3Ø892  LETZTE ZUFALLSZAHL (2BYTES)
78AE 3Ø894  FLAG: Ø=SUCHE VARIABLE, 1=NEUE VAR. EINTRAGEN
78AF 3Ø895  TYP-FLAG FÜR VARIABLE IM ACCU WRA1
            2=INTEGER 3=STRING 4=SNG.PREC 8=DBL.PREC
78BØ 3Ø896  ZWISCHENWERT WÄHREND BEARBEITUNG EINES AUSDRUCKS
78B1 3Ø897  ADRESSE HÖCHSTE RAM-ZELLE (MEM SIZE)
78B3 3Ø899  ZEIGER AUF NÄCHSTEN FREIEN PLATZ IN LSPT
78B5 3Ø9Ø1  LSPT (LITERAL STRING POOL TABLE)
78D2 3Ø93Ø  ENDE LSPT

78D3 3Ø931  LÄNGE & ADRESSE EINES STRINGS, DER ZUM STRINGBE-
            REICH TRANSPORTIERT WIRD
78D6 3Ø934  ZEIGER AUF NÄCHSTEN VERFÜGBAREN RAUM IM STRING-
            BEREICH
78D8 3Ø936  ZEIGER AUF LETZTES BYTE DER AKTUELLEN ANWEISUNG/
            EDIT-FLAG WÄHREND PRINT USING
78DA 3Ø938  ZEILENNR. DES LETZTEN GELESENEN DATA-STATEMENTS
78DC 3Ø94Ø  'FOR'-FLAG: 9 FOR-LOOP AKT. Ø=NICHT AKTIV
78DD 3Ø941  FLAG INPUT: Ø=KEYBOARD SONST READ
78DE 3Ø942  READ FLAG: Ø=RAED-ANW. AKTIV 1=INPUT-ANW. AKTIV
            BEI USING: TRENNER ZWISCHEN STRING & VARIABLE
78DF 3Ø943  PROGR.START-ADR. BEI LOADING UNTER DOS
78E1 3Ø945  AUTO-FLAG: Ø=KEIN AUTO  <>Ø=AUTO
78E2 3Ø946  AKTUELLE ZEILENNR. BEI INPUT
78E4 3Ø948  reserviert
```

```
78E6 3Ø95Ø  BEI INPUT: ADRESSE DER AKTUELLEN ZEILE
            BEI RUN: ZEILENNR. DER AKTUELLEN ZEILE
78E8 3Ø952  ZEIGER FüR BASIC-STACK
78EA 3Ø954  ZEILE, IN DER LETZTER FEHLER AUFTRAT (ERL)
78EC 3Ø956  ADRESSE DES BEFEHLS, BEI DEM ERROR AUFTRAT
78FØ 3Ø96Ø  ANFANGSADR. DER ON ERROR ROUTINE
78F2 3Ø962  ERROR FLAG: FEHLER SETZT 255, RESUME SETZT Ø
78F3 3Ø963  ADR. DES DEZIMAL-PUNKTES IM BUFFER
78F5 3Ø965  LETZTE AKT. ZEILENNR., VON END,STOP GERETTET
78F7 3Ø967  ADR. LETZTES BYTE BEARBEITET BEI ERROR
78F9 3Ø969  ZEIGER ENDE BASIC/ BEGINN VARIABLEN
78FB 3Ø971  ZEIGER ENDE VARIABLEN/BEGINN FELD-VARIABLEN
78FD 3Ø973  ZEIGER (ENDE FELD)/BEGINN FREE SPACE
78FF 3Ø975  ZEIGT AUF ZEICHEN NACH LETZTEM BEI READ
            GELESENEN ZEICHEN
79Ø1 3Ø977  DEFAULT-TABELLE VARIABLEN-TYP-BEZEICHNUNGEN
            26 BYTES (A-Z)
            BEINHALTET EIN VARIABLENNAME KEINEN TYPCODE
            WIRD ER ASU DER TAB. ENTNOMMEN. NEW TRÄGT Ø4 EIN
            ÄNDERUNG DURCH BASIC-UP DEFINT,DEFSNG,DEFDBL
791B 31ØØ3  TRACE-FLAG: 175=TRACE ON  Ø=TRACE OFF
791C 31ØØ4  CARRY FüT SCHIEBE-OPERATIONEN
791D 31ØØ5  X-REGISTER * WRA1 * LSB DBL. PREC
791E 31ØØ6  WRA1 DBL.PREC WERT
791F 31ØØ7  WRA1 DBL.PREC WERT
792Ø 31ØØ8  WRA1 DBL.PREC WERT
7921 31ØØ9  WRA1 LSB INTEGER / SNG.PREC
7922 31Ø1Ø  WRA1
7923 31Ø11  WRA1 MSB SINGLE PREC
7924 31Ø12  WRA1 EXPONENT SNG.PREC
7925 31Ø13  VORZEICHEN DES ERGEB. BEI MATH. & ARITHM. OPERAT
7926 31Ø14  ZWISCHENWERT BEI DBL.PREC ADDITION
7927 31Ø15  Y-REGISTER * WRA2 * LSB
792D 31Ø21  WRA2 MSB

792E 31Ø22  WRA2 EXPONENT

792F 31Ø23  nicht benutzt

793Ø 31Ø24  INTERNER PRINT-BUFFER
7949 31Ø49  LETZTES BYTE PRINT-BUFFER
794A 31Ø5Ø  ZWISCHENSPEICHER DBL.PREC DIVISION (DIVISOR)
```

------------- EIGENE EINTRAGUNGEN --------------------------

Anhang 13: Erweiterter ASCII-Code, Bildschirm-Code

POKE	ASCII	CHAR	POKE	ASCII	CHAR	POKE	ASCII	CHAR	POKE	ASCII	CHAR	L110	L210
32	32	SPACE	33	33	!	34	34	"	35	35	#		
36	36	$	37	37	%	38	38	&	39	39	'		
40	40	(	41	41	)	42	42	*	43	43	+		
44	44	,	45	45	–	46	46	.	47	47	/		
48	48	0	49	49	1	50	50	2	51	51	3		
52	52	4	53	53	5	54	54	6	55	55	7		
56	56	8	57	57	9	58	58	:	59	59	;		
60	60	<	61	61	=	62	62	>	63	63	?		
0	64	@	1	65	A	2	66	B	3	67	C		
4	68	D	5	69	E	6	70	F	7	71	G		
8	72	H	9	73	I	70	74	J	11	75	K		
12	76	L	13	77	M	14	78	N	15	79	O		
16	80	P	17	81	Q	18	82	R	19	83	S		
20	84	T	21	85	U	22	86	V	23	87	W		
24	88	X	25	89	Y	26	90	Z	27	91	[		
28	92	\	29	93	]	30	94	^	31	95	←		
32	96	SPACE	33	97	!	34	98	"	35	99	#		
36	100	$	37	101	%	38	102	&	39	103	"		
40	104	(	41	105	)	42	106	*	43	107	+		
44	108	,	45	109	–	46	110	.	47	111	/		
48	112	0	49	113	1	50	114	2	51	115	3		
52	116	4	52	117	5	54	118	6	55	119	7		
56	120	8	57	121	9	58	122	:	59	123	;		
60	124	<	67	125	=	62	126	>	63	127	RUBOUT		
128	128	□	129	129	▗	130	130	▖	131	131	▄	h'grau	grün
132	132	▝	133	133	▐	134	134	▞	135	135	▟		
136	136	▘	137	137	▚	138	138	▌	139	139	▙		
140	140	▀	141	141	▜	142	142	▛	143	143	█		
144	144	□	145	145	▗	146	146	▖	147	147	▄	weiß	gelb
148	148	▝	149	149	▐	150	150	▞	151	151	▟		
152	152	▘	153	153	▚	154	154	▌	155	155	▙		
156	156	▀	157	157	▜	158	158	▛	151	159	█		
160	160	□	161	161	▗	162	162	▖	163	163	▄	d'grau	blau
164	164	▝	165	165	▐	166	166	▞	167	167	▟		
168	168	▘	169	169	▚	170	170	▌	171	171	▙		
172	172	▀	173	173	▜	174	174	▛	175	175	█		
176	176	□	177	177	▗	178	178	▖	179	179	▄	d'grau	rot
180	180	▝	181	181	▐	182	182	▞	183	183	▟		
184	184	▘	185	185	▚	186	186	▌	187	187	▙		
188	188	▀	189	189	▜	190	190	▛	191	191	█		
64	192	@	65	193	A	66	194	B	67	195	C		
68	196	D	69	197	E	70	198	F	71	199	G		
72	200	H	73	201	I	74	202	J	75	203	K		
76	204	L	77	205	M	76	206	N	79	207	O		
80	208	P	81	209	Q	82	210	R	83	211	S		
84	212	T	85	213	U	86	214	V	87	215	W		
88	216	X	89	217	Y	90	218	Z	91	219	[		
92	220	\	92	221	]	94	222	↑	95	223	←		
96	224	□	97	225	!	98	226	"	99	227	#		
100	228	$	101	229	%	102	230	&	103	231	"		
104	232	(	105	233	)	106	234	*	107	235	+		
108	236	,	109	237	–	110	238	.	111	239	/		
112	240	0	113	241	1	114	242	2	115	243	3		
116	244	4	117	245	5	118	246	6	119	247	7		
118	248	8	121	249	9	122	250	:	123	251	;		
124	252	<	125	253	=	126	254	>	127	255	?		

KONTROLL-KODE TABELLE

ASCII	FUNKTION	ASCII	FUNKTION	ASCII	FUNKTION	ASCII	FUNKTION
0	CR/LF	1		2		3	
4		5		6		7	
8	CURSOR LINKS	9	CURSOR RECHTS	10	CURSOR ABW. LF	11	
12		13	CR/LF	14		15	
16		17		18		19	
20		21	INSERT	22		23	
24	CURSOR LINKS	25	CURSOR RECHTS	26		27	CURSOR AUFW.
28	CURSOR HOME	29	CR	30		31	CLEAR SCREEN

127 RUBOUT

TABELLE DER NUR MIT POKE DARSTELLBAREN ZEICHEN

POKE	CHAR	POKE	CHAR	POKE	CHAR	POKE	CHAR
192	□	193	▗	194	▖	195	▄
196	▝	197	▐	198	▞	199	▟
200	▘	201	▚	202	▌	203	▙
204	▀	205	▜	206	▛	207	█
208	□	209	▗	210	▖	211	▄
212	▝	213	▐	214	▞	215	▟
216	▘	217	▚	218	▌	219	▙
220	▀	221	▜	222	▛	223	█
224	□	225	▗	226	▖	227	▄
228	▝	229	▐	230	▞	231	▟
232	▘	233	▚	234	▌	235	▙
236	▀	237	▜	238	▛	239	█
240	□	241	▗	242	▖	243	▄
244	▝	245	▐	246	▞	247	▟
248	▘	249	▚	250	▌	251	▙
252	▀	253	▜	254	▛	255	█

Anhang 14: Geometrische Funktionen

Funktion	BASIC-Formulierung
SECANT	SEC(X)=1/COS(X)
COSECANT	CSC(X)=1/SIN(X)
COTANGENT	COT(X)=1/TAN(X)
INVERSE SINE	ARCSIN(X)=ATN(X/SQR(-X*X+1))
INVERSE COSINE	ARCCOS(X)=-ATN (X/SQR(-X*X+1))+pi/2
INVERSE SECANT	ARCSEC(X)=ATN(X/SQR(X*X-1))
INVERSE COSECANT	ARCCSC(X)=ATN(X/SQR(X*X-1)) +(SGN(X)-1)*pi/2
INVERSE COTANGENT	ARCOT(X)=ATN(X)+pi/2
HYPERBOLIC SINE	SINE(X)=(EXP(X)-EXP(-X))/2
HYPERBOLIC COSINE	COSH(X)=(EXP(X)+EXP(-X))/2
HYPERBOLIC TANGENT	TANH(X)=EXP(-X)/EXP(X)+EXP(-X))*2+1
HYPERBOLIC SECANT	SECH(X)=2/(EXP(X)+EXP(-X)
HYPERBOLIC COSECANT	CSCH(X)=2/(EXP(X)-EXP(-X))
HYPERBOLIC COTANGENT	COTH(X)=EXP(-X)/(EXP(X)-EXP(-X)*2+1
INVERSE HYPERBOLIC SINE	ARCSINH(X)=LOG(X+SQR(X*X+1)
INVERSE HYPERBOLIC COSINE	ARCCOSH(X)=LOG(X+SQR(X*X-1)
INVERSE HYPERBOLIC TANGENT	ARCTANH(X)=LOG((1+X)/(1-X))/2
INVERSE HYPERBOLIC SECANT	ARCSECH(X)=LOG((SQR(-X*X+1)+1)/X)
INVERSE HYPERBOLIC COSECANT	ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1)/X
INVERSE HYPERBOLIC COTANGENT	ARCCOTH(X)=LOG((X+1)/(X-1))/2

Anhang 15: Kurzschreibweisen (Shorthands)

```
ANWEISUNG  ZEICHEN  BESCHREIBUNG
----------------------------------------

PRINT      ?        KANN IM BASIC-TEXT BENUTZT WERDEN,
                    WIRD ABER ALS 'PRINT' GELISTET

REM        '        ERSETZT 'REM'

LET        =        STATT LET A=5 KANN A=5 GESCHRIEBEN
                    WERDEN

THEN       ,        KANN VIELFACH ENTFALLEN ODER DURCH
                    KOMMA ERSETZT WERDEN, Z.B.
                      IF A=Ø GOSUB 1000
                      IF A<>B PRINT "XXXX"
                    ABER:
                      IF A>5 , 200

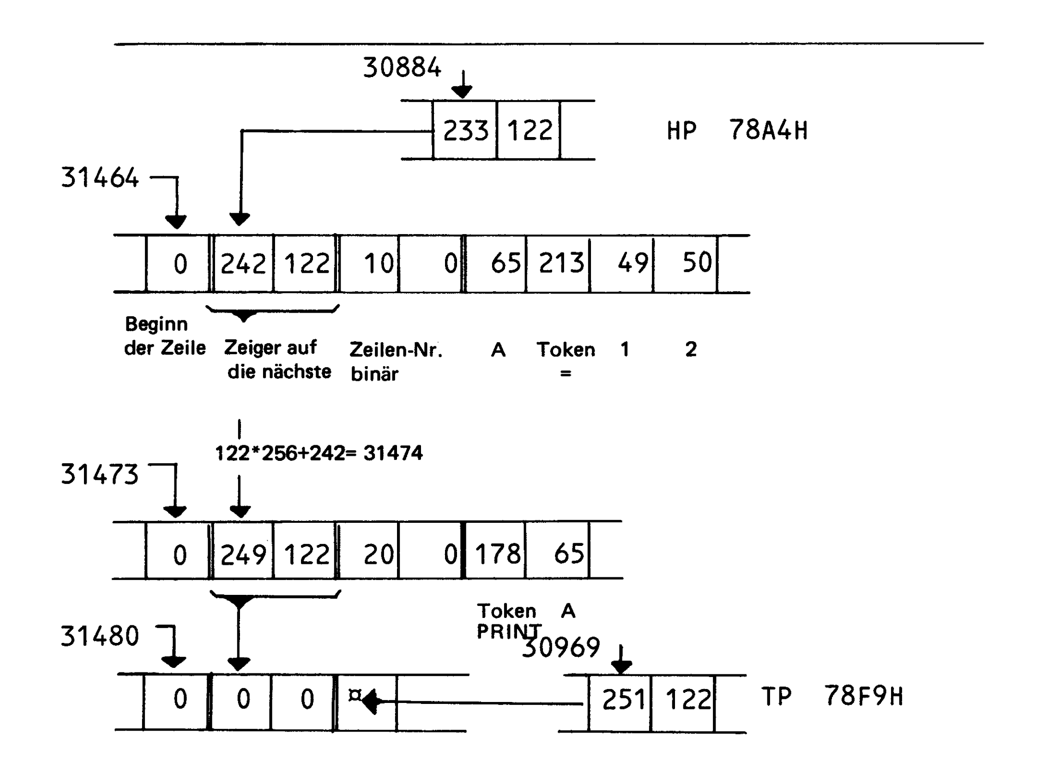
           ;        KANN ENTFALLEN, WENN EINE EINWAND-
                    FREIE TRENNUNG DER AUSZUGEBENDEN
                    VARIABLEN UND KONSTANTEN VORLIEGT,
                    Z.B. PRINT A$ B$ "SEMIKOLON"
                    ABER: PRINT A;B

GOTO                KANN NACH THEN ENTFALLEN, Z.B.
                    IF A>5 THEN 200
```

Anhang 16: BASIC Text-Format

```
10 A = 12
20 PRINT A

FOR I=0 TO 18:?PEEK(31464+I);:NEXT <RETURN>
```



Anhang 17 : BASIC Tokens

128	END	129	FOR	130	RESET
131	SET	132	CLS	133	*
134	*	135	NEXT	136	DATA
137	INPUT	138	DIM	139	READ
140	LET	141	GOTO	142	RUN
143	IF	144	RESTORE	145	GOSUB
146	RETURN	147	REM	148	STOP
149	ELSE	150	COPY	151	COLOR
152	VERIFY	153	<DEFINT>	154	<DEFSGN>
155	<DEFDBL>	156	CRUN	157	MODE
158	SOUND <ERROR>	159	<RESUME>	160	OUT
161	<ON>	162	*	163	*
164	*	165	*	166	*
167	*	168	*	169	*
170	*	171	*	172	*
173	*	174	<SYSTEM>	175	LPRINT
176	*	177	POKE	178	PRINT
179	CONT	180	LIST	181	LLIST
182	<DELETE>	183	<AUTO>	184	CLEAR
185	CLOAD	186	CSAVE	187	NEW
188	TAB(	189	TO	190	*
191	USING	192	<VARPTR>	193	USR
194	<ERL>	195	<ERR>	196	*
197	*	198	POINT	199	*
200	<MEM>	201	INKEY$	202	THEN
203	NOT	204	STEP	205	'+'
206	'-'	207	'*'	208	'/'
209	' '	210	AND	211	OR
212	'>'	213	'='	214	'<'
215	SGN	216	INT	217	ABS
218	<FRE>	219	INP	220	*
221	SQR	222	RND	223	LOG
224	EXP	225	COS	226	SIN
227	TAN	228	ATN	229	PEEK
230	*	-------------------------		242	*
243	LEN	244	STR$	245	VAL
246	ASC	247	CHR$	248	LEFT$
249	RIGHT$	250	MID$	251	*
252	*	-------------------------		255	*

*   NICHT BELEGTER CODE

< XX > TOKEN WIRD VON DER 'BASIC UP'-ERWEITERUNG ERKANNT.

Anhang 18: Tape Loading Format

	T: Text File	B: Binary File	D: Data File
SYNC. Bytes	255 Bytes of 80H	255 Bytes of 80H	255 Bytes of 80H
HEADER	5 Bytes of FEH	5 Bytes of FEH	5 Bytes of FEH
EXTENSION	1 Byte of F0H	1 Byte of F1H	1 Byte of F2H
FILENAME	16 Bytes (max.) of ASCII	16 Bytes (max.) of ASCII	16 Bytes (max.) of ASCII
GAP	3 ms Blank	3 ms Blank	3 ms Blank
START ADDRESS	2 Bytes of binary	2 Bytes of binary	----
END ADDRESS	2 Bytes of binary	2 Bytes of binary	----
Program Content	xx Bytes	xx Bytes	----
Data Content	----	----	xx Bytes
Checksum	2 Bytes	2 Bytes	2 Bytes
End of File Marker (EOF)	20 Bytes of Zeroes (00H)	20 Bytes of Zeroes	----
Terminator	----	----	1 Byte of 00H

Anhang 19

Vergleich LASER-BASIC mit verschiedenen BASIC-Dialekten
Hinweise zum Umschreiben veröffentlichter Programme

1. LASER-BASIC-Ergänzung durch "EXTENDED BASIC":

ON A GOSUB 100,200,...	DEFINT	VARPTR
ON A GOTO 100,200,...	DEFSNG	POS (X)
FRE (0)	DEFDBL	RANDOM
FRE ("")	DEFSTR	
ON ERROR GOTO	RECT	
ERL	CIRCLE	
ERR	PAINT	
RESUME	MEMSIZE = HIMEM	
LPEN	PLOT ... TO ...	

2. Ton-Kommandos

Anweisung	**LASER-BASIC**
500 BEEP	500 SOUND 12,1
500 CLICK	500 POKE 30862,80: POKE 30863,52 501 X=USR(0)

3. FOR ... NEXT-Schleifen

FOR I=0 TO V — wird einmal durchlaufen, wenn V=0 wird. In anderen BASIC-Versionen werden die Anweisungen zwischen FOR -- NEXT dann nicht ausgeführt.

4. Variablen-Namen

Namen, länger als zwei Zeichen, sind im LASER-BASIC zulässig. Es werden jedoch nur die ersten beiden Zeichen gewertet. Z.B. verwendet TI99/4A 16 Zeichen, im SINCLAIR-BASIC werden sogar beliebig lange Namen verwendet.

Beispiel:

```
10 WERT=5
20 WELT=12
30 PRINT WERT;WELT
```

Ein Programmlauf ergibt bei den genannten Rechnern 5 und 12 als Ergebnis. Das LASER-BASIC gibt jedoch 12 und 12 aus, da zweimal die Variable WE angesprochen wurde. Eine Anpassung der Variablen-Namen ist ggfs. vorzunehmen.

5. Druckersteuerung

Die Anweisungen

```
200 OPEN 4,4,0
400 PRINT#4,"TEXT"
490 CLOSE4
```

sind gegen

400 LPRINT''TEXT''

auszutauschen, die OPEN- und CLOSE-Anweisungen zu streichen.

6. Zufallszahlen mit RND (X)

In einigen BASIC-Versionen erzeugt RND (X) Zufallszahlen im Bereich 0 bis 1. Nachfolgende Anweisungen rechnen diese Zahlen in den gewünschten Bereich um. Alle diese Anweisungen müssen entfallen, im LASER-BASIC gilt als Argument X die Obergrenze der zu erzeugenden Zufallszahl.

A=RND (49)

erzeugt eine Zahl im Bereich 0-49.

7. String-Verarbeitung

```
BASIC-DIALEKTE           LASER-BASIC
--------------------------------------
SEG$(A$,1,4)             LEFT$(A$,1,4)
A$(TO 4)                 LEFT$(A$,A)
A$ (4 TO)                RIGHT$(A$,4)
SEG$(A$,6,5)             MID$(A$,6,5)
A$(6 TO 10)              MID$(A$,6,5)
A$(7)                    MID$(A$,7,1)
```

8. Zeichen von der Tastatur übernehmen

```
10 GET A$                A$=INKEY$:A$=LEFT$(A$,1)
10 CALL KEY              DITO
```

9. PRINT-Anweisungen

```
100 PRINTI;SPC(I);I*2    100 PRINT I;
                         110 FOR J=0 TO I:PRINT" ";:NEXTI
                         120 PRINT I*2

100 PRINT "  "           100 CLS    ;BILDSCHIRM LOESCHEN
100 PRING "  "           100 PRINT CHR$(28) ;CURSOR HOME

100 INVERS               100 POKE    :REM INVERS ON
                         100 POKE    :REM INVERS OFF
```

10 Bildschirmaufbau

In den meisten Fällen ist eine Neugestaltung des Bildschirmaufbaues notwendig, abhängig vom Format des Bildes in Zeilen und Spalten. Die gezielte Ausgabe auf den Bildschirm PRINT AT X,Y; wird im LASER-BASIC zu:

```
Z=32*X+Y
PRINT@Z,
```

wobei X und Y die Bildschirm-Koordinaten und Z die direkte Nummer der Bildschirmstelle ist. Dabei ist zu beachten, daß einige BASIC-Interpreter die erste Bildschirmposition mit 0 entspr. X=0 und Y=0 bezeichnen, während andere jeweils 1 einsetzen.

Die relativen Cursor-Steuerbefehle bei VC- und CBM-Computern sind auf den LASER nur mit zeilenlangen PRINTCHR$(xx);CHR$(xx) ...-Anweisungen zu übertragen. Ein neuer Bildschirmaufbau wäre sinnvoller.

11. Genauigkeit der Zahlendarstellungen

Siehe Kapitel 1: Variablen doppelter Genauigkeit.



Software-System, Handbuch I